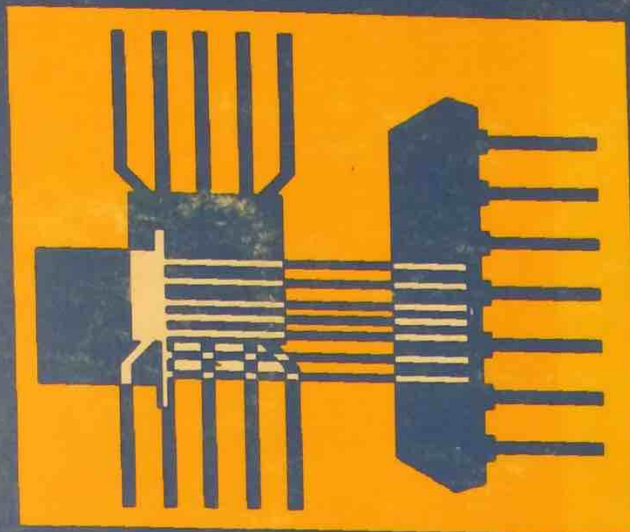


microprocessors

NEW DIRECTIONS



FOR DESIGNERS

Edited by Edward A. Torrero

Selected from ELECTRONIC DESIGN

TK
7874
T67
1975

HAYDEN

MICROPROCESSORS
New Directions for Designers

Selected from
ELECTRONIC DESIGN

MICROPROCESSORS

New Directions for Designers

Selected from
ELECTRONIC DESIGN

Edited by
EDWARD A. TORRERO
Associate Editor, *Electronic Design*



HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey

SANGAMON STATE UNIV. LIBRARY
SPRINGFIELD, ILLINOIS

Library of Congress Catalog Card Number: 75-21855

Copyright © 1975 by HAYDEN BOOK COMPANY, INC. All rights reserved. No part of this book may be reprinted, or reproduced, or utilized in any form or by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying and recording, or in any information storage and retrieval system, without permission in writing from the Publisher.

Printed in the United States of America

3	4	5	6	7	8	9	PRINTING	
76	77	78	79	80	81	82	83	YEAR

Preface

Digital system designers now have a remarkable new tool. Microprocessors can lower the cost and increase the flexibility of electronic equipment, thereby blazing a trail for new and exciting uses.

Applications once confined largely to point-of-sale-terminals now run the gamut from traffic-control systems to small accounting equipment, and from computer terminals to industrial-process controllers. In some cases, microprocessor-based systems have even replaced dedicated minicomputers.

Together with memory and peripheral circuitry, processor chips form complete microcomputers. In complexity, these micros fall midway between small, hand-held calculators and conventional minicomputers. And microcomputers bring with them some of the features of both. Like calculators, they're compact and inexpensive. But like minicomputers they can be programmed for a wide range of tasks and work with such computer peripheral devices as magnetic memories and high-speed printers.

Through imaginative design, engineers can use software to adapt basic microprocessor hardware for a host of applications. However, the software phase of design can entail an extensive learning process for those engineers who are more at home with gates and registers.

This book deals with both the hardware and software aspects of microcomputer design. It presents application articles, tutorial discussions, and survey reports published in *ELECTRONIC DESIGN* from 1973 to 1975. The emphasis is a practical one. Subjects covered in detail include how to select circuits, how to interpret their capabilities, how to extend their useful range, and how to apply them.

July 1975

EDWARD A. TORRERO

Contents

Section I. Industry Enters the Microprocessor Era

Focus on Microprocessors	3
Microprocessors in Test Equipment: Promises and Delays	20
Smart Machines in Industrial Electronics	26
Micro vs Mini: Mini Manufacturers Counterpunch	28

Section II. Getting Started: Microprocessor Basics

MOS/LSI Microprocessor Selection	33
Analysis of Microprocessor Instruction Sets	37
MOS/LSI Microcomputer Coding	45
Traffic-Light Controller: An Example	51
Microcomputer I/O Capabilities	56
Microprocessor or Random Logic?	62
The Anatomy of a Microprocessor Chip	67
Clearing the Interface and Software Hurdles	73

Section III. Grappling with Microprocessor Limitations

Speeding Microprocessor Multiplication	81
Microprogramming Extends LSI-Processor Capabilities	89
Debug that Microcomputer System with a Mini	95
Improving Microprocessor Interrupt-Handling Capability	99
Providing the Clock and Drive Signal for the 8080	101

Section IV. Applying Microprocessors

Printer Control	105
Simplifying Add-On Peripheral Controllers	115
PLL for Motor Control Uses Microprocessor	121
Microprocessor IC's Improve Instruments	125
Micros Join Intelligent Networks	130

SECTION I:

Industry Enters the Microprocessor Era

The first article discusses the wide range of products available and explores the pitfalls of microprocessor selection. Other articles deal with microprocessor penetration into instrumentation and industrial electronics, and its threat to minicomputers.

Focus on Microprocessors	3
<i>Edward A. Torrero, Associate Editor, Electronic Design</i>	
Microprocessors in Test Equipment: Promises and Delays	20
<i>Stanley Runyon, Associate Editor, Electronic Design</i>	
Smart Machines in Industrial Electronics	26
<i>John F. Mason, Associate Editor, Electronic Design</i>	
Micro vs Mini: Mini Manufacturers Counterpunch	28
<i>John F. Mason, Associate Editor, Electronic Design</i>	

Focus on Microprocessors

EDWARD A. TORRERO

Associate Editor, Electronic Design

Digital designers faced with the many performance claims for new LSI microprocessors might conclude that one of these little, low-cost marvels could solve all their system problems. After all, if a small IC offers the processing of a computer, replaces scores of standard logic circuits, and has a seemingly endless list of applications, who needs to design with anything else?

Closer examination of the burgeoning application literature for a "computer on a chip" reveals a different picture. A large-scale integration processor, for example, does perform many of the functions of the central processing unit in conventional computers. But to use the circuit, many more ICs may be needed to interface with peripheral devices, data-communication lines and even its own memory.

And that low price tag on the LSI processor—typically, well under \$100 for 4-bit word-length units and under \$400 for 8-bit units—is low compared with the thousands of dollars needed for a general-purpose minicomputer. However, to build a full-fledged microcomputer, you need memory, and the cost of that can easily exceed the price of the processor.

Moreover, if you add up the cost of all the necessary hardware components, you may find the total exceeding the less-than-\$1000 level of "stripped down" minicomputers available on PC boards.

Programming—unfamiliar demands

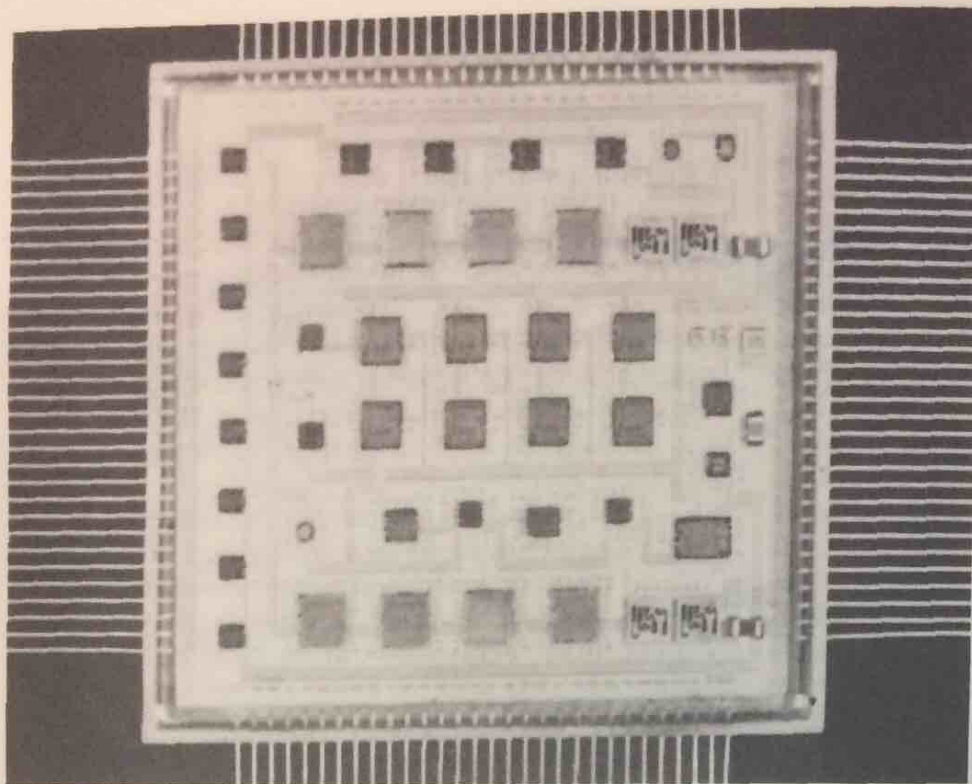
A decision to use an LSI processor opens up a whole new design ball game. Not only must a digital designer contend with the relatively familiar requirements of any logic system. The



Combine an LSI microprocessor with associated circuitry and memory, and you obtain a microcomputer. In the foreground are LSI circuits and prototype systems offered by Rockwell International. The manufacturer's line includes 4 and 8-bit microprocessor chip sets.

designer must also grapple with the relatively unfamiliar demands of programming. And software development, usually involving assembly language (only one step up from a computer's inherent machine language of ONEs and ZEROs), represents by far the major design effort and cost.

In addition microprocessors don't lend themselves to the traditional "learning curve" for new



Dice mounted on a substrate form a microprocessor system. This unique packaging approach is used by Teledyne Semiconductor. Other vendors typically mount DIPs on PC boards.

components. Previously the experience gained in the use of one component could be transferred to a similar component from another manufacturer. But different microprocessors generally don't have alternate sources. And the chips come with different software capabilities, hardware requirements and design aids. Hence a completed design, using one LSI processor, might have to be scrapped totally if you decide to turn to another vendor's unit.

Still, there are major benefits. Microprocessors are unique among ICs in that they can be programmed like computers. As a result, they permit a tradeoff of software for hardware to achieve a dazzling increase in system capability and versatility.

They can be used economically to replace or upgrade random-logic designs, involving scores of standard ICs, when many functions must be performed. And they use less circuitry than hard-wired logic in applications emphasizing random collection and routing of data.

Of course, for some applications microprocessors are not the sole LSI alternative. Complex logic decisions might be handled just as well with circuitry using programmable logic arrays. And arithmetic computations are performed by arithmetic logic units or by calculator chips—from which a number of microprocessors have evolved. Custom LSI chips provide yet another alternative.

Nevertheless microprocessors are filling the gap between special-purpose LSI circuits and conventional minicomputers. They are currently finding their greatest use as decentralized,

cheaper minis in remote, programmable controllers.

There's a wide product range

The many advantages of microprocessors are creating a demand that manufacturers are meeting in a variety of ways. Designers can choose from among single or multichip processors, chip sets or PC-board assemblies, and from among a wide range of technologies.

The most common technology is silicon-gate, p-channel MOS (PMOS). But manufacturers are also using n-channel MOS (NMOS) and silicon-on-sapphire MOS (SOS/MOS) to achieve speeds that are higher than those possible with PMOS. Bipolar processes are employed for the highest speeds—about 200-ns cycle vs 2 μ s for NMOS types. Complementary MOS (CMOS) is used for the lowest power dissipations—microwatt-range chip dissipation vs milliwatt range for other types.

Standard LSI-processor products in various configurations handle data in 4, 8 and 16-bit word lengths, and modular multichip microprocessors can be used to achieve even longer word-length processing. The available configurations, with their major features, consist of the following:

- Microprocessor chip sets, including special interface ICs and sometimes special memories, to simplify designs of minimum-hardware systems for specific applications.
- Microprocessor-based logic boards to eliminate the need to test, assemble and interconnect

processor chips, peripheral circuits and memory for a variety of applications.

- General-purpose—non dedicated—microcomputers, on cards or in boxes, to permit system design, development and testing. These are offered by component manufacturers, as well as, a growing number of other vendors.

- Microprocessor-based minicomputers offering, as a result of their traditional mini features, maximum flexibility and capability when compared with MOS microcomputers. Offered by minicomputer manufacturers, these units generally use custom MOS processors.

The cost of each configuration increases with a unit's complexity. At the minicomputer level, hardware cost might be the highest. But available software support is the most extensive. Designers tend to feel that the number of units determines the major tradeoff in a choice between a micro and minicomputer. A small number of units—as for an end-user application—can best be served by a mini, which can minimize software development costs. But for large quantities—as for an OEM application—a microcomputer can minimize hardware costs.

Chip sets improve early versions

Due to increasing availability of special interface circuits and improvements in processor-chip architecture, fewer additional circuits are needed for the newest microprocessors than for earlier versions. First-generation 8-bit processors, for example, typically needed about 20 additional standard-TTL circuits to make them work.

The extra ICs include the following: registers to address memory, either ROM or RAM; decoders to interface with memories; other ICs to handle processor information and to synchronize the operation of the processor and circuits; clock circuits, and a variable amount of interface ICs, depending on the application. For example, in a multichannel data-communications application each channel requires an asynchronous receiver/transmitter and associated interface ICs.

But even with newer processors, applications still can require additional circuitry to obtain one or more of the following: clock generation and timing, memory and I/O control, data and address buffering, multiplexed inputs, interrupt control, refresh for dynamic memories and additional supply voltages.

And some microprocessors are offered only as part of complete chip sets or with the purchase of the associated memory from the same IC manufacturer. This may not be a problem if you plan to buy all your components from the same source, but it does stop you from shopping around for the lowest price and precludes the use of core memory—which most processors can ac-

cept just as well as semiconductor types. Also, if you purchase a chip set, you must design around the circuits offered.

Specs don't tell all

Unlike the less-complicated ICs, microprocessors cannot be completely characterized on a simple data sheet. Moreover different vendors use different parameters to measure a processor's capabilities. This makes any comparison of processors—not to mention selection of the best one—a difficult task. And there's no trend in sight toward standardization.

A microprocessor's computing speed is a case in point. Frequently manufacturers use a basic cycle time, or period—sometimes called a microcycle—to denote speed. But many microcomputer operations require several such cycles to be performed. This applies especially to the execution of the more powerful instructions. Hence a critical instruction may require more time than that indicated by the basic cycle.

In addition a microprocessor's maximum clock rate can be misleading, if taken for a measure of speed. It's possible for one microprocessor to perform basic operations—like register-to-register add—faster than a unit using a higher clock rate. Differences in microprocessor architecture and chip design tend to minimize the importance of the clock-rate spec.

Other specs given to indicate speed include minimum instruction time, interrupt response time, and time to add two numbers—which may already reside in the processor. Like cycle time and clock rate, these numbers don't measure such critical times as the over-all time needed to perform important routines. Excluded are additional delays, such as those needed to obtain data from memory. The solution is to use benchmark programs tailored to your application as a basis for selection of one microprocessor over another.

Use of benchmark programs can determine the power of the instruction set, thus circumventing one of the most abused specs—the number of instructions. Microprocessor comparisons based on this number abound, although such comparisons have serious flaws.

For example, a simple number doesn't reveal what instructions are available for data movement and manipulation, for decision and control and for input/output operations. Some microprocessors have far more I/O instructions than others; they are tailored for a specific class of applications. And missing instructions can always be performed by routines, although with a sacrifice in speed.

Also, the number of instructions claimed for the same microprocessor can increase from one page of a reference manual to another. One

reason is that instructions that move data have been multiplied by the number of addressing modes.

Common addressing modes include direct, immediate and indirect. In the immediate mode, the instruction includes data, while in the indirect mode, an address preloaded into a register increases the address bits in an instruction. Variations and extensions of these modes are also available, so a basic instruction can be multiplied several times.

Other factors that inflate the number of instructions may be the number of registers—in, say, a load-to-register operation—or the number of conditions—for example, those on which a branch may occur.

Of course, improved instruction sets are obtained with longer word-length microprocessors and advanced versions of smaller units. For example, 16-bit microprocessors have instructions for multiply and divide—functions that require

models require external components to achieve the interface levels needed for logic compatibility.

Watch architectural claims, too

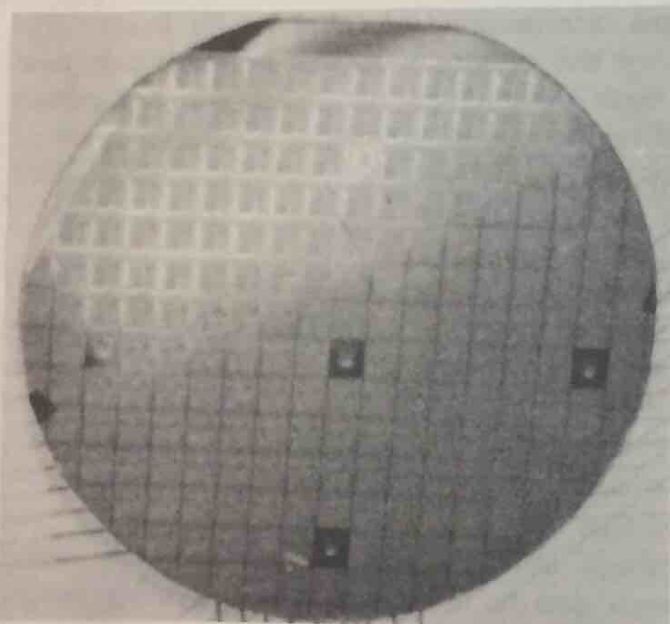
Many computer-like features of microprocessors are frequently cited, including the number and function of registers, the type and depth of stack, interrupt capability and direct-memory access (DMA). However, there isn't as much architectural diversity as there is with minicomputers. IC manufacturers are constrained by technology limitations, so that comparable microprocessors tend to perform similarly. For example, preliminary benchmark programs run by several manufacturers for their 8-bit, single-chip NMOS processors often show comparable execution times.

Data sheets frequently boast several working registers. But only a single register, the accumulator, is essential. An accumulator, however, must have access to memory, and available instructions should permit immediate addressing and data manipulation between the accumulator and memory. If indirect addressing is available, even the function of special index registers can be accomplished with memory.

The major significance of additional registers lies in access time and the bit efficiency of instruction words. It takes far fewer bits to specify one of several previously defined working registers than a memory location. And a faster execution time can be obtained with registers that are separate from memory. They can be accessed without excessive memory-cycle delays. Otherwise it doesn't matter whether these registers are in an external memory or in the processor, so long as they can be referenced efficiently.

Some data sheets might seem to imply that the quantity of registers is more significant than their quality. It isn't. Not all registers can be incremented and tested for zero, even though they are described as "general-purpose." Those that can may be used for counting and program-loop control. In general, not all registers can be used for indexed addressing. Nor can they be loaded directly from memory—rather than only from the accumulator—or used as a source or destination for arithmetic logic operations.

Microprocessors employ stack-oriented registers that can be accessed only in a last-in-first-out basis—the so-called LIFO, or push-down, stack. These are used for subroutine nesting, interrupts and for temporary storage of data. They can be either on the chip (a hardware stack) or external to the processor in memory (a software, or pointer, stack). The hardware stack permits higher-speed operation, but it has limited size. The size of the software stack may be as large as available memory space permits, but the stack must be maintained by the program.



They're not "computers on a chip"—yet. But LSI microprocessors, symbolized by this wafer from Intel, perform many of the functions of central-processing units in conventional computers.

software routines in 8-bit models.

And advanced microprocessors feature more powerful instructions as well as the original set of a predecessor. However, this "software compatibility" doesn't allow routine upgrading of systems by chip replacement. In general, expect to redesign to employ the new hardware/software tradeoffs efficiently.

For virtually all models, data sheets claim TTL compatibility. But don't expect many MOS processors to drive TTL loads; most don't. The term primarily refers to the fact that both microprocessor and TTL circuit can use a common 5-V supply. Newer models list a maximum TTL-drive capability of only one standard load, and many

An interrupt capability is an absolute must for applications that involve asynchronous or unpredictable events. All microprocessors claim some type of interrupt handling ability, but the extent can vary from one unit to another. With older processors, you have to design the means to save the contents of the processor just prior to the interrupt, and then restore the information after the interrupt is serviced.

Those means may involve reservation of registers on the chip, use of external registers or use of another microprocessor. Any of these methods can store the essential contents of the processing unit. Software control, involving special routines, must also be provided to complete the design. The complexity of these techniques tends to discourage designers from attempting to handle even a single interrupt.

Newer microprocessors can accommodate single-line, multilevel and vectored interrupts, and they save essential registers automatically. A complete saving must be programmed. In one single-line interrupt system, device-interrupt requests are ORed together to form one request line. The program identifies the device and resolves priority. A multilevel scheme employs several single-level sense lines to handle additional interrupts. For very fast response, the vectored interrupt directly branches to a memory location that corresponds to a specific interrupt.

Another feature that depends on the unit is DMA capability. For some units, a "direct" access of memory must be performed indirectly through the microprocessor's usual word-by-word transfer procedure. This may not be a problem if you don't mind the processor's idle time, but it does limit data-transfer efficiency.

And don't expect I/O data throughput rates always to include the time needed to sense for a device or to respond to an interrupt. When these times are included, the actual throughput can be significantly less than expected.

Fixed instruction vs microprogram

Most microprocessors come with fixed instruction sets, around which software must be developed for an application. For some units, however, the option exists for microprogramming, the ability to alter or totally change the original instruction set. In essence, you program the microprocessor's internal microinstructions to obtain a macroinstruction set that is tailored to the application.

The advantages of microprogramming include increased speed, since microinstructions are executed considerably faster than macroinstructions are. Also, the technique allows a more detailed level of control that can be used to reduce hardware; the program controls more functions.

Because of the hardware savings, vendors expect microprogramming to find its greatest use in large-volume applications—in excess of tens of thousands of units. Alternatively, microprogramming represents the logical choice for an emulation of another computing system or for the speedy execution of critical, short routines.

The exceptional skills required for microprogramming constitute its major disadvantage. A microprogrammer must deal with the specific timing relationships of the internal architecture. And since each application requires a separate microprogram, each has its own instruction set that can't be transferred easily to another application. Nor can software design aids, geared toward the fixed instruction set, be applied to the changed set.

Design aids speed development

Much of the start-up, or development, effort in the design of a microcomputer system is linked to the coding phase. Coding converts system programs into instructions that can be loaded directly into the memory. However, the basic design-aid tools themselves are programs that generally require the use of time-sharing services or other computer facilities. And like the LSI processors they support, design-aid features can differ from vendor to vendor.

Assemblers are a case in point. All assemblers convert a program into the basic machine language in a process that usually involves several steps. Essentially the assembler reads a so-called source tape—with statements written in the mnemonic, or symbolic, assembly language—and produces a so-called object tape, with binary numbers suitable for the processor's memory. Errors due to misuse of the assembly language can be detected and pointed out by the assembler.

But some manufacturers offer single-pass assemblers, thereby reducing the steps needed to obtain the binary instructions for memory. Or they may provide the option of loading the assembler into ROMs and pROMs so that the microcomputer itself, rather than a host computer, executes the program. These are called hardware assemblers.

Another type, called a macro-assembler, simplifies coding when similar sections of code are used repeatedly, but variations preclude the use of conventional subroutine techniques. With a macro-assembler, a single instruction yields the necessary expansion.

Editors, available on time-sharing services, allow designers to prepare the original assembly-language programs and to change or correct them with simple commands. They can add documentation and store, combine and retrieve programs. And they can readily output programs onto paper

μP Scorecard®	Classification	Technology	Parts Family								Features	Word Size (Data/Program)	Address Capacity (Program Words)	Clock (kHz/Phases)	Register Add Time (μsec per Data Word)	Number of CPU Registers			Return-Stack Size (NR x Bits)	Voltages Required	Power Dissipation (Watts)	Operating Temperature Range (°C)	Package Sizes (DIP Pins)	Price Range (approx. CPU only; Quantity 100)	Status	Remarks	
			Clock Driver	I/O Interface	UART/USRT	RAM	ROM/PROM	Interface	Interrupts	Integrated CPU						Microprogrammed	Accessible Stack	DMA Ability									BCD Arithmetic
BURROUGHS MINI-D	One-chip CPU with ROM	PMOS													8/12	256	1000/1	9	3	—	1	—	—	16	\$ 60	Custom	
FAIRCHILD PPS-25	Calculator-oriented	PMOS													4x25/12		400/2	62.5	1	—	—	—	4x12	16, 18, 24, 40	\$ 60	Delivered	
INTEL MCS-4/4004	Calculator-oriented	PMOS													4/8	4K	740/2	10.8	1	—	16	—	4x12	16	\$ 30	Stocked	
INTEL 4014	Calculator-oriented	PMOS													4/8	8K	740/2	10.8	1	—	24	—	8x12	16, 24		Rumored	
INTEL MCS-8/8008	One-chip CPU	PMOS													8/8	16K	500/2	20	1	—	6	—	8x14	18	\$100	Stocked	
INTEL 8008-1	One-chip CPU	PMOS													8/8	16K	800/2	12.5	1	—	6	—	8x14	18	\$130	Stocked	
INTEL 8080	One-chip CPU	NMOS													8/8	64K	2083/2	2	1	—	6	—	(RAM)	40	\$200	Delivered	
INTERSIL ISD-8	One-chip CPU	CMOS													12/12	4K	2000/1	6	1	—	—	—	Modifies Program	40		Announced	DEC PDP-8 Code
MOTOROLA 6800	One-chip CPU	NMOS													8/8	64K	1000/2	2	2	1	—	—	(RAM)	24, 40	\$150	Samples	
NATIONAL GPC/P	4-bit Slice	PMOS													4N/23	100	715/4	1.4	8	—	—	—	16x4N	22, 24	\$150	Delivered	1 ≤ N ≤ 6
NATIONAL IMP-4	3-chip CPU	PMOS													4/4	64K	500/4	12	4	—	—	—	7x12	22, 24	\$150	Samples	16x4 Data Stack
NATIONAL IMP-8	3-chip CPU	PMOS													8/8	64K	715/4	4.6	3	1	—	—	16x8	22, 24	\$230	Delivered	
NATIONAL IMP-16	5-chip CPU	PMOS													16/16	64K	715/4	4.6	2	2	—	—	16x16	22, 24	\$310	Delivered	
RAYTHEON RP-1600	4-bit Slice	Bipolar													4N/48	64K	5000/1	1	j	k	—	—	—	22, 24		Announced	(j + k) ≤ 8
RCA COSMAC	2-chip CPU	CMOS													8/8	64K	667/1	6	p	q	—	—	1x16	40		Announced	(p + r + 2q) ≤ 15
ROCKWELL PPS-4	Calculator-oriented	PMOS													4/8	4K	200/2	5	1	—	1	—	3x12	42	\$ 40	Delivered	
ROCKWELL PPS-8	One-chip CPU	PMOS													8/8	16K	256/2	12	1	1	2	—	2x14	42		Announced	
SIGNETICS 2650	One-chip CPU	NMOS													8/8	32K	1200/1	4.8	s	t	—	—	8x15	40	\$100	Announced	(s + t) ≤ 6
TOSHIBA TLC8-12	One-chip CPU	NMOS													12/12	4K	1000/3	13	1	—	—	—	(RAM)	16, 24, 26, 42		Announced	Multiply Instruction

July 1, 1974

July 1, 1974

A tabulation of representative LSI microprocessors reflects the product range: from calculator-oriented, 4-bit units using PMOS technology to newer 16-bit multichip bipolar processors. Note that the table, reprinted from "New Logic Notebook" by Microcomputer Techniques,

© Microcomputer Technique, Inc.

compares push-down stacks—called return stacks—in terms of number of registers (NR) and the bit size of each. Also processor registers are divided into accumulators (ALU), index registers (XR) and remaining registers (GP) for general use.

tape as well as printers.

A number of loaders are available to complete the coding process. With these, which can be stored in ROMs, assembled programs are loaded into read-only memory. They can also be loaded into RAMs, in which case a bootstrap type is used. A so-called relocating loader automatically adjusts program addresses and loads the resulting instructions. And some loaders have linking capability that lets you use routines with undefined labels. These types supply the missing cross-references between separate routines.

Several manufacturers also offer compilers, which allow programs to be written in a high-level language. The benefits are many: A short readable compiler statement corresponds to many symbolic assembly-language statements. Compilers eliminate the need to write detailed codes to control loops, to access complex data structures or to program formulas and functions. And since programming details are lessened, errors are reduced.

But while high-level language programs are compact, easy to read and much easier to write, the net result could be excessive storage space and slower execution, when compared with an assembly-language program. Generally a choice between the two approaches depends on the degree of optimization required and the design time allowable.

In addition to these design aids, test programs—such as simulators—are virtually mandatory to track down the various subtle errors that may remain. Similarly hardware prototype units are essential to the development of the final product. Prototype units generally involve expanded memory capability, teletypewriter or card-reader interface, power supply, chassis and control panel—in addition to a microcomputer.

Besides boosting initial development costs for the designer, the wide range of hardware/software support requires a major investment by the semiconductor manufacturers. This investment is in addition to that needed to produce the LSI chips. In fact, one indication of a vendor's seriousness in marketing a particular microprocessor is the availability of hardware/software support for the product.

Still, ever more manufacturers are entering the field because of the high potential payoff. Various sources predict that the microcomputer market—valued at under \$50-million last year—should reach at least \$500-million in four years. In the process, a sizable chunk of the TTL market could be replaced. Major microprocessor-chip vendors—such as Intel, National Semiconductor and Rockwell International—are meeting the challenge in a variety of ways.

The Components

By any standard, the recognized leader in microprocessors is Intel, which introduced the product in 1971. Benefiting from its early, one-to-two-year lead over competitors, Intel reportedly captured as much as three-quarters of last year's microcomputer market. Moreover each of its processor chips has been a first of its kind, beginning with a 4-bit unit, the 4004, and leading to an 8-bit PMOS model, the 8008, and the latest advance, an 8-bit NMOS microprocessor—the 8080.

Among the Intel products, the 8080 sets the pace for increased speed and improved instructions. The silicon-gate processor has a 2- μ s instruction cycle and 74 basic instructions, which include the 48 instructions of the earlier 8008. The additional 30 instructions and a 6:1 faster execution rate provide up to a 10:1 speed advantage over the 8008. Moreover the improved performance of the 8080 is obtained with a typical power dissipation of only 600 mW, the same as that of the 8008.

The 8080 can address up to 65-k bytes of memory without need for an external address register. This compares with 16-k bytes of memory and an external register for the 8008. The 8080 requires only six peripheral ICs, as contrasted with the 20 needed with the 8008. The NMOS 8080 comes in a 40-pin package and operates from +12 and ± 5 -V supplies.

A number of architectural differences account for the improved performance of the 8080. For example, it contains a 16-bit stack pointer (to operate the external LIFO stack) and a 16-bit program counter (to indicate the next instruction), instead of an address storage stack with eight 14-bit locations. A portion of the external memory can be used as a last-in, first-out stack, addressed by the stack pointer upon the execution of a Call, Return or Restart instruction.

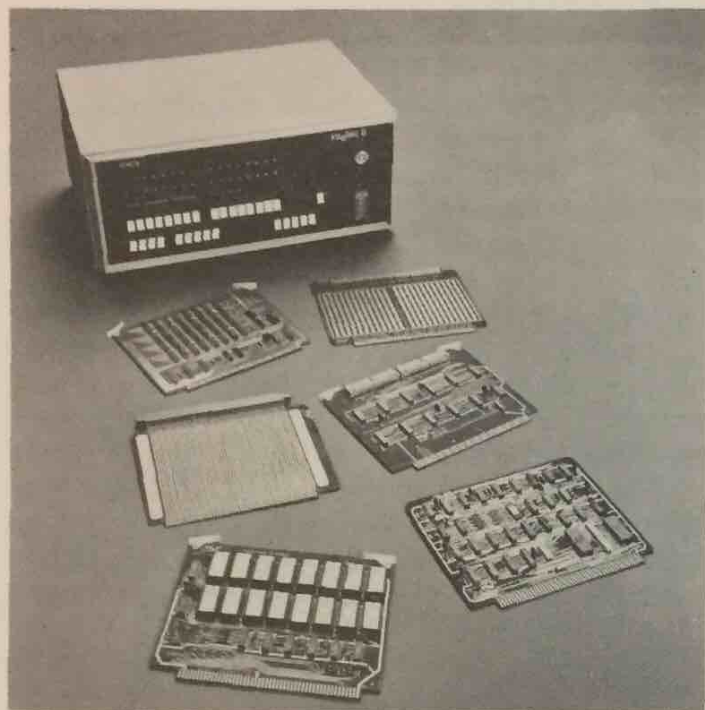
Moreover not only the program counter but also the data register, the accumulator and the flags (bits that are set to indicate various conditions) can be saved in the external push-down stack. As a result, multiple interrupts can be handled more easily with the 8080.

The 8080 can perform BCD and binary arithmetic. It also has capability for double-precision arithmetic involving two 16-bit numbers. The NMOS processor can handle up to 256 input ports and a similar range of outputs.

Intel offers several hardware and software design aids for the development of microcomputer

systems with its processors. The aids include the Intellec series, consisting of expandable, modular systems that come complete with microprocessor, memory, power supplies and circuitry for teletypewriter interface and clock generation. Each Intellec system is housed in a compact cabinet that features a control and display panel for immediate system monitoring and debugging. All program storage can be accomplished with RAMs, rather than ROMs, for easier program loading and modification. After a program is firm, it can be loaded into ROM.

The standard software package for the Intellec



Most manufacturers offer design aids to simplify development of microcomputer systems. Intel's aids include the Intellec series, which comes complete with memory, supplies and clock and interface circuitry.

series includes a system monitor, contained in pROMs, resident assembler and text editor. A programming module provides the timing and level shifting to program pROMs. Additional support is provided by a cross-assembler and simulator written in Fortran IV and by a PL/M compiler available on time-sharing terminals.

The use of PL/M, derived from IBM's PL/1 language, permits sample programs to be written in a fraction of the time needed to write the same program in assembly language. PL/M offers a far simpler means of programming, compared with assembly language. And the debugging and checkout times of a PL/M program are less, because the structure of the language allows the compiler to detect error conditions that would not be spotted by an assembler.

Recent additions to the Intel product line include several ICs, intended to simplify system

design and expansion, and an advanced version of the company's 4-bit microprocessor. The additional circuits include a 4-k-bit dynamic RAM (the 8107), a 2048 \times 8-bit ROM (the 8316), a 512 \times 8-bit pROM (the 8604), and several peripheral circuits. The latter interface communications lines, handle increased loads and replace IC packages now required.

Intel's new 4-bit microprocessor, the 4014, features software compatibility with the earlier 4004. Additional instructions permit logic operations, such as AND and OR. The 4014 also allows storage capacity of 8-k bytes of memory, increased from the 4-k bytes for the 4004. Other capabilities include improved single-interrupt handling and a single-step mode of operation to simplify testing and debugging.

IMP series—chips, cards and 'boxes'

National Semiconductor offers a broad line of PMOS microprocessor products. They consist of 4, 8 and 16-bit parallel processors that are available in chip form, on card subsystems and in complete microcomputer boxes. Each microprocessor features downward software compatibility—it's compatible with a microprocessor having a shorter word length, but not one with a longer word length. And the fixed instruction set of each can be altered or changed through microprogramming techniques.

The National microprocessors are built around two building-block chips: a Register and Arithmetic Logic Unit (RALU) and a Control Read-Only Memory (CROM). The RALU is a 4-bit "slice"; four are used with one or two CROMs to obtain the 16-bit system, and eight RALUs with two CROMs can be used to form a 32-bit system. The CROM provides storage for the manufacturer's fixed instruction set and the control logic for up to eight RALUs.

The IMP-16C, National Semiconductor's 16-bit microprocessor system, comes on an 8-1/2 \times 11-in. PC board. It consists of the processor, clock system, I/O bus drivers, 256 words of RAM and provisions for 512 words of ROM or pROM memory.

The IMP-16C uses a basic 43-instruction set and an expanded 17-instruction set provided by a second CROM. The additional 17 speed processing with instructions that include divide, multiply and double-precision add and subtract. The basic microcycle, or machine cycle, is 1.4 μ s. Several microcycles are needed to execute a typical instruction. Two 16-bit numbers can be multiplied for a 32-bit result in a speedy 150 μ s.

Each RALU supplies the IMP-16C with an accumulator and a push-down stack. A total of

four accumulators improve the bit efficiency of instruction words by cutting down memory-cycle delays. The hardware stack permits rapid nesting of subroutines and interrupts, and its limited depth can be extended—for, say, overflow conditions—by use of main memory.

The microprocessor also features vectored as well as slower, single-line interrupts. With a single-line interrupt, the total overhead time to get to the service routine can be as high as 34.85 μ s, since it depends on the number of peripheral devices, and these might number 16. But with a vectored interrupt, the total overhead is only 4.55 μ s, a figure that doesn't change with the number of devices.

Several software design aids are offered with each microprocessor. Programs are available for cross and self-assembling, source editing, debugging, and absolute and relocate loading. Also, driver/utility and diagnostic aids are offered. Hardware design aids using assembly language come as complete microcomputers and prototype systems. For the 16-bit system, these are the IMP-16L and 16P boxes. Both units contain a 16-bit microprocessor card.

The IMP-16L has a front-panel display, which provides access to memory and microprocessor registers. The box comes with 4-k, 16-bit words, expandable to 65-k words. Also, a high-speed asynchronous bus permits direct-memory access by peripheral devices without the need to go through the RALUs.

The IMP-16P can interface with a teletypewriter for application software development. This prototyping tool comes complete with chassis, control panel, power supplies and one or more 4-k, 16-bit-word read/write memory modules. National Semiconductor says the IMP-16P is the box to start with to begin a 16-bit design.

Recent additions to National Semiconductor's line are an advanced 4-bit microprocessor system and a microprogramming tool to help designers alter or change the fixed instruction set. The 4-bit system performs BCD arithmetic and uses one CROM, one RALU and a Four-bit Interface Logic Unit, called FILU. The interface unit combines the functions of a number of standard-TTL peripheral circuits.

The microprogramming tool, called Field-Alterable Control Element (FACE), makes use of the fact that a change of CROMs changes the fixed instruction set. FACE, which comes on a card, replaces the CROM on a microprocessor board to form what mainframe designers call writable-control store. Microprocessor control logic now becomes accessible to external ROM or RAM for development of a tailored microprogram. The final result can be stored in a custom-masked CROM or a bipolar pROM. The increased speed of the bipolar memory can reduce delays

incurred through connection of external, off-the-board circuitry.

PPS family provides chip sets

Aiming to reduce the number of additional components often needed with single-chip microprocessors, the Microelectronics Div. of Rockwell International offers chip sets for both 4 and 8-bit systems. The sets come complete with processor, clock circuit, memory and input/output ICs. For additional flexibility, you can get a combination ROM and RAM chip or you can select from a growing number of interface and peripheral circuits.

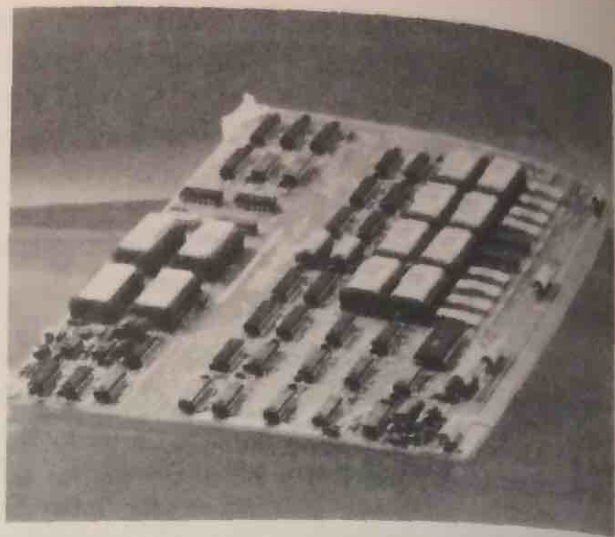
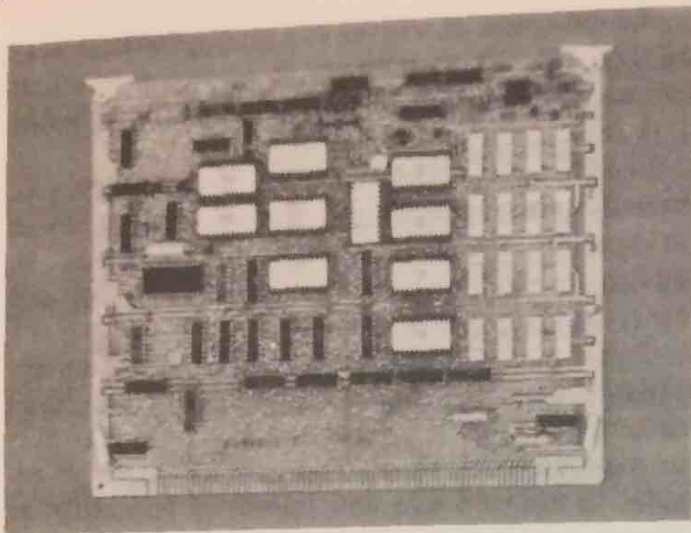
Rockwell estimates that up to 30 peripheral circuits can be eliminated by use of the PPS-4 or PPS-8 chip sets. These are 4 and 8-bit parallel processing systems, respectively. Moreover some of the special circuits in the Rockwell chip sets are not generally available with competing processors. For example, a nonvolatile RAM is being developed for the PPS-4 set. And a controller circuit, in an advanced development stage, for the PPS-8 will permit direct memory access. Rockwell is also developing other circuits for use with peripherals such as CRTs, printers and floppy discs.

The PPS-8 features over 90 instructions, with capabilities for decimal and binary arithmetic single-byte subroutine call and digit/byte manipulation. A complete instruction cycle can be executed in 4 μ s, and decimal addition or subtraction is performed at 12 μ s per digit. Among 4-bit microprocessors, the PPS-4 sets the pace for speed, with an instruction cycle of 5 μ s and a register-to-register add time of 2.5 μ s. The 50 instructions for the PPS-4 contain logic and conditional and unconditional data-transfer operations.

The PPS chip sets use a relatively slow clock, typically 200 kHz. However, signals internal to the system are handled at four times that rate. The clock generator provides the processor with two synchronized signals, which are then divided logically into four phases, thus boosting the speed.

Bus lines transfer data during the second and fourth time intervals. In the alternate intervals, address and data-bus lines are automatically cleared to zero. This interface timing scheme permits direct connection of an extensive number of circuits. For the PPS-4, up to 30 circuits can share the bus without need for additional buffering or drive circuitry.

Also, the control logic within the processor allows arithmetic or logic instructions to be carried out in one cycle time. Addition of two decimal digits requires six instructions, or six cycle times. Hence for the 5- μ s cycle of the PPS-4, two decimal



Processor cards provide an assembled and tested microprocessor system, complete with memory. Clockwise from the top left are National Semiconductor's 16, 8 and 4-bit models (the IMP-16C/300, 8C/200 and IMP-4, respectively). National units also feature a microprogramming option that can be used to alter or change the fixed instruction set.

digits can be added or subtracted in 30 μ s.

Software aids consist of cross-assemblers and simulators available on time-shared services. The aids may also be purchased for use on in-house computers. Hardware aids range from Assemblators (developed by Applied Computing Technology) to evaluation boards, containing microprocessors, RAMs, I/O circuits and clocks.

Micro Systems

Aiming to shorten the usual development time for a microcomputer design, some manufacturers are offering systems that constitute full-fledged microcomputers and complete prototype units. These vendors use chips supplied by Intel, National Semiconductor or Rockwell International.

Product forms range from consoles with circuitry on conventional PC boards to LSI dice mounted on a single substrate. Besides providing the necessary components for a host of applications, these products also feature additional circuitry that helps extend the capabilities of the internal LSI microprocessor.

An early microcomputer entry is the Micral series from R2E (Realisations Etudes Electroniques, based in France). Micral systems, built around Intel's 8-bit microprocessors, come

in three processing speeds: 12 μ s (basic model), 6 μ s (Micral G) and 2 μ s (Micral S).

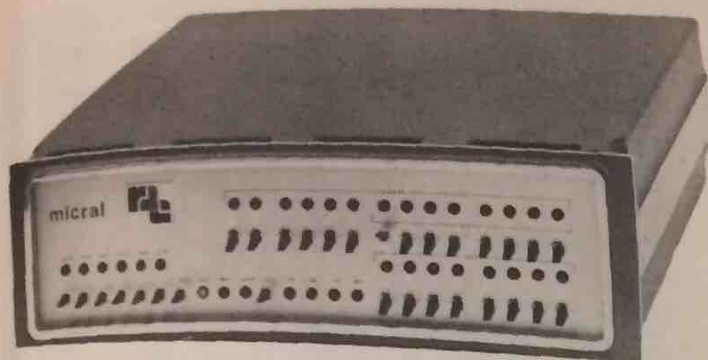
Although Micral uses the 8008, it has several instructions beyond those available with the 8008. The additional instructions permit improved handling of interrupts and allow register saves. Interrupt-driven I/O channels can be operated at a maximum throughput of 56-Mbyte/sec. Also, the Micral G (and S) includes five added instructions for data manipulation.

Software support includes a two-pass assembler, keyboard editor and diagnostics for peripherals and I/O interfaces supplied with the microcomputer. In addition R2E offers a high-level control language to simplify prototype development.

Micral units have a memory capacity of 16-k bytes, and this can be increased to 64 k. The direct addressing capacity of peripherals is a hefty 1792 bytes at the input and 23 at the output. Up to seven I/O channels can be operated.

Applied Computing Technology provides a series of Assemblators—short for assemblers/simulators—for the Intel and Rockwell 4-bit processors. Assemblators, which can interface directly with a teletypewriter, reduce the development phase of microcomputer designs.

In a typical development phase, a designer specifies and interconnects the I/O circuitry, which is then interfaced to the Assembler. Programs are written and assembled in the Assembler, which holds the utility system in ROM



Complete microcomputers extend the capabilities of the basic microprocessor used. R2E's Micral system, for example, provides additional instructions for interrupt handling and data manipulation. R2E uses Intel chips.

and the assembled program in a special emulation ROM. As a result, testing of programs involves simple manipulation of Assembler console switches or the teletypewriter, so that program debugging can be reduced to a fraction of that normally required. And because the unit contains a resident assembler, costly time-sharing changes can be eliminated.

Both the CBC-4, the Assembler for the Intel chip, and the PPS-4MP, which is used for Rockwell's model, are similar. However, the CBC-4 features a single-pass assembler for simplified assembly. For both units, the corrected program can be recorded in pROMs, which are then inserted into the microcomputer to complete the design.

Pro-Log features the PLS-400 family of microcomputers on $4\frac{1}{2} \times 6\frac{1}{2}$ -in. PC cards. Built around Intel's 4-bit processor, the 4004, the PLS-400 also contains Intel's 4002 RAMs and 1702A erasable pROMs. A minimum system consists of a single card with 1024 bytes of memory and 32 TTL I/O lines. For expanded capability, a three-card system contains 4096 bytes of memory and 128 TTL I/O lines.

Also available from Pro-Log is the MPS-800 family of 8-bit microcomputers for data processing. The family consists of three-card and five-card systems. Both units come with 256 words of pROM and 1024 words of RAM data/instruction memory. The memory can be expanded to 16-k words.

Process Computer Systems offers a microcomputer built around Intel's 8080 microprocessor. Called the Series 2000 Micro CPU, the system aims for such industrial applications as torque monitoring and control and also engine testing.

Plug-in modules allow the Series 2000 to control a variety of processes. The I/O modules include a 12-bit a/d converter, 16-bit digital input interface and modules for relay outputs. These units communicate with the system's memory and microprocessor through a common back-

plane data bus. Moreover the microprocessor sees all I/O devices as memory locations, precluding the need for special—and possibly less efficient—I/O instructions.

Process Computer Systems also offers plug-in breadboard cards, which allow the development of special analog and digital I/O modules. In addition assembler, compiler, debugging and other programs are available.

Teledyne Systems takes a unique packaging approach for its microcomputers. Rather than mount DIPs on PC boards, it mounts IC dice on ceramic substrates. The hybrid approach results in reduced size and improved reliability at prices that are comparable with PC-board equivalents. A single package measures only $2 \times 2 \times 0.2$ -in.

Teledyne's microcomputers are the TDY-52A, which uses Intel's 4004 microprocessor, and the TDY-52B, which uses National Semiconductor's dice to form a 16-bit unit. Instructions for the TDY-52A can be tailored to an application by alteration of the control microprogram in ROMs mounted on the substrate.

The TDY-52A requires external power supplies and some I/O circuitry, as does the 16-bit unit. In addition the TDY-52B requires an external clock (5.7 MHz square wave) and memory (ROM, RAM or combinations of both). Memory configurations are available in hybrid packages identical to those of the microcomputers.

New LSI Processors

At various stages of introduction—from chip development to some product delivery—are a multitude of LSI processors, all of which are expected to be available in quantity by about the middle of next year. The great majority use NMOS technology, handle 8-bit word lengths and have a basic cycle of about $2 \mu s$, as does Intel's 8080. And most of these are being offered with special peripheral circuits and memories to minimize the circuitry otherwise needed.

In addition to these circuits, manufacturers are developing 12-bit processors that employ CMOS technology and emulate popular minis. And bipolar/LSI processors, featuring high-density techniques, promise to close the speed gap between microcomputers and conventional minis.

Five ICs comprise the basic microcomputer chip set promised by Motorola in its M6800 family. The heart of the set is the microprocessing unit, or MPU—an 8-bit parallel processor. The MPU lists 72 instructions and seven ad-

dressing modes, with one mode reserved for two 8-bit accumulators on the chip. It also uses 16-bit memory addressing, allowing memory size up to 65-k addresses.

The MPU shares data and address buses with special byte-organized memories—1024 × 8-bit ROMs and static 128 × 8-bit RAMs—and two programmable I/O circuits, the Peripheral Interface Adapter (PIA) and the Asynchronous Communications Interface Adapter (ACIA). The special I/O circuits permit reduced interface circuitry.

All five silicon-gate NMOS chips operate from a single 5-V supply. A total of 10 circuits of the M6800 family can be interconnected on the bus without reduction of the maximum clock rate of 1 MHz, which must be supplied by an external clock.

The interface circuits look like memory locations to the MPU. Hence the MPU can use the same instructions for, say, peripherals, connected to the PIA that it uses with RAM; special I/O instructions are not needed. The PIA controls and transfers data and status information to and from peripheral devices or, possibly, other microprocessors.

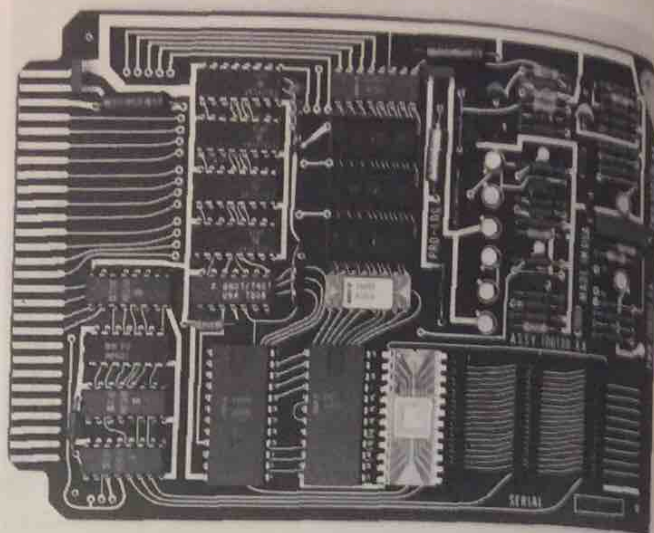
Much of the control logic for peripherals can be handled by the PIA. In an application described by Motorola, for example, a PIA replaces the 16 to 20 standard MSI logic circuits usually required to control popular OEM printers. The complete interface consists of the PIA, peripheral drivers and comparators. The latter provide TTL-compatible logic levels in the face of inductive surges from the printer.

Similarly the ACIA performs all the functions of a standard universal asynchronous receiver/transmitter. It relieves the MPU of many of the timing and control functions of asynchronous data communications. And it can interface directly with standard modems, including a special modem IC planned for the M6800 family.

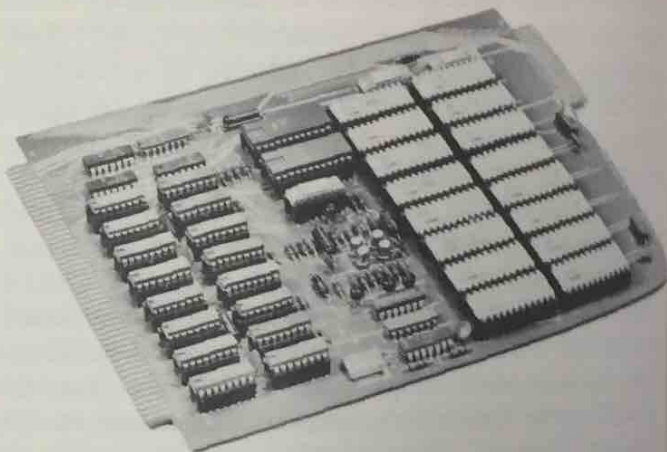
The MPU features decimal and binary arithmetic capabilities, variable-length instructions and double-byte operations. Addressing modes include extended, indexed, implied and relative, as well as immediate and direct. The minimum execution time, as for the Add instruction, is 2 μ s. The maximum time reaches 12 μ s in the case of a Software Interrupt instruction.

A special feature of Electronic Arrays' 8-bit NMOS parallel processor is string or series capability. Up to 16 bytes of data can be entered from memory or from an I/O device, and stored in the chip's multiregister accumulator for processing. And the 16-byte data transfer can be accomplished with a single instruction.

Moreover the chip contains a 24-byte RAM, called the Program-Address Storage (PAS). It can be used as a push-down stack, for interrupt



Microcomputers on PC boards are offered by Pro-Log. Models available include a 4-bit system for dedicated controller applications and an 8-bit model for data processing. Both systems use Intel chips.



Prototype systems help cut development time. Applied Computing Technology calls its systems Assemblators, or combination assemblers/simulators. The vendor supplies prototype units for microprocessors offered by Intel and Rockwell International.

vectors, for program-call storage, or for a combination of these functions. For expanded capability, the PAS can be extended into external RAM.

The 8-bit processor has about 66 instructions and addresses up to 65-k bytes of memory. The four addressing modes are direct, immediate, indexed and extended. Memory can be ROMs or RAMs of various speeds in any combination.

Other circuits planned by Electronic Arrays for the microprocessor consist of the following: a 1024 × 8-bit ROM, 256 × 4-bit RAM and an I/O controller. The I/O chip is intended to be a general-purpose, programmable logic design for any interface.

Fairchild Semiconductor plans to offer the F8 chip set, consisting of an 8-bit NMOS Isoplanar microprocessor, a memory interface chip, an 8-k bit ROM and a custom RAM. The system features a basic cycle of 2 μ s and has internal clock and interface circuitry. Direct-drive capability exists

for standard peripherals, such as teletypewriters and printers.

Fairchild estimates that improvements in the F8 family can reduce the amount of peripheral circuits by 10 to 30%, compared with other 8-bit NMOS processors. Also, preliminary benchmark programs indicate speed improvements by a factor of 1-1/2 to 3.

Software support includes a compiler to accept the PL/M language and a special conversational utility system that requires an interface module. The system allows a program to proceed in slow motion, so designers can debug a program with paper tape or cassette. The utility system avoids the need to modify the contents of a pROM each time a correction is required.

An earlier microprocessor from Fairchild is the PPS-25 system, a 4-bit chip set. Intended for scientific calculators and control systems, the PPS-25 uses standard ICs for interfacing of keyboards and displays. The PPS-25 specs an add time of 125 μ s for two groups of 16 digits.

The PIP: 8 addressing modes

Signetics' upcoming Programmable Integrated Processor (PIP) allows the use of eight addressing modes, for increased flexibility and efficiency in programming. Operand addresses for instructions can be defined in these modes: register, immediate, relative, absolute, indirect and indexed. Branch addresses may be formed through the use of a relative or absolute mode, and each may be direct or indirect.

The PIP responds to over 64 instructions, the most complex of which are executed in less than 10 μ s. Instructions feature variable lengths for improved bit efficiency; they may be 1, 2 or 3 bytes long. Also each of the chip's four general-purpose registers may be used as an accumulator. Hence processing bottlenecks that result from having only a single accumulator are eliminated.

The Signetics' chip operates from a single 5-V supply. Its input clock frequency is variable down to dc; externally the PIP resembles a static IC. The processor's address lines allow direct addressing for up to 32-k bytes of memory. And PIP's vectored-interrupt capability allows indirect branching to any location in memory.

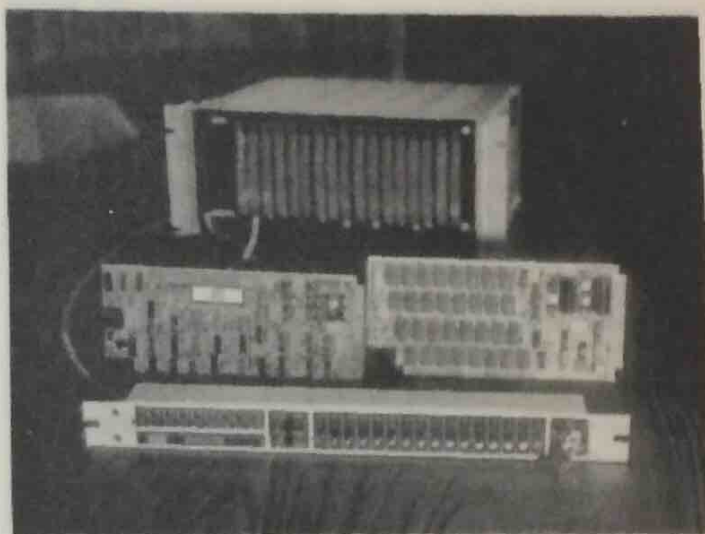
A full range of design aids is planned for the PIP, including a cross-assembler written in Fortran II rather than the more common but advanced Fortran IV. The use of the more basic language permits the assembler program to be run on a wide range of computers, from mainframes to minicomputers.

In a rare example of microprocessor alternate sourcing, Microsystems International Ltd. offers the 8008, the 8-bit PMOS processor introduced by Intel. MIL provides the microprocessor, as

well as the various components needed to make an 8-bit microcomputer, on seven basic boards. Each board measures 4-1/2 x 6-in. and uses a standard dual 22-way edge connector.

On the MOD8-1 board, for example, are the microprocessor, clock generators and state-decoding and bus-switching control logic. An I/O board, the MOD8-2, contains teletypewriter interface and reader-control and system-restart logic. Other boards provide ROM, RAM, buffer and I/O circuitry.

MIL's design aids include the Monitor-8 software package, which allows interactive debugging of designers' programs entered in assembly language from a teletypewriter. In



Microcomputers aim for industrial applications such as engine testing and torque control. Process Computer System's Series 2000 Micro CPU features a common backplane data bus on which interface modules communicate with the microprocessor and memory.

addition the company offers a similar applications package for a series of boards that will use a proprietary 4-bit microprocessor expected shortly. The 4-bit system is called the MOD4 series.

Proving that PMOS is still around for 8-bit units, Mostek is sampling an 8-bit p-channel, parallel processor called the MK5065. Compared with first-generation units, the MK5065 expands the number of instructions and slashes the number of peripheral circuits. Also, the chip's architecture is three-leveled for rapid handling of interrupts. One or two of the chip's three program counters and three accumulators, for example, can be reserved for immediate servicing of interrupts.

Bipolar processors emerge

The first general-purpose bipolar/LSI microprocessor has just been announced by Raytheon

Semiconductor. A seven-chip processor called the RP16, the 16-bit unit surpasses MOS microprocessors in speed through the use of bipolar technology, increased addressing modes and an improved instruction set. Two versions of the RP16 are slated. Model A emphasizes arithmetic capabilities, while Model B stresses byte handling.

While most microprocessor chips are intended for random-logic designers, a Schottky-TTL processor from Monolithic Memories is aimed at the computer architect. The special-purpose microprocessor, dubbed the Model 6701 microcontroller, can be used to emulate conventional computers and replace scores of packages at reduced power. A single 6701 can perform the function of 25 TTL MSI packages while saving about 6 W.

The 6701 consists of four 4-bit controllers and associated ROMs and shift registers. None of the ROMs is used to decode the processor's 36 instructions, which are at the fundamental level of arithmetic logic units. As a result, microprogramming techniques can be employed to apply the IC.

A single cycle takes about 200 ns. But within this period the 6701 can execute such functions as subtract, shift and store. The bipolar speeds assure a precise emulation of conventional machines, which use standard-bipolar circuits. Moreover the 6701 can be expanded in increments of 4 bits, so that other than 16-bit systems can be designed without significant reductions in speed. Other applications foreseen for the 6701 include high-speed peripheral controllers and point-of-sale systems.

12-bit models fill a gap

Although much of the current microprocessor activity involves 8 and 16-bit units, two manufacturers—Intersil and Toshiba—are filling the gap with 12-bit models. The Intersil processor is designed to be a CMOS/LSI equivalent of Digital Equipment Corp.'s popular PDP-8 mini-computer. The Toshiba processor (the TLCS-12) is a proprietary PMOS design, with such component-saving features as a timing generator on the chip.

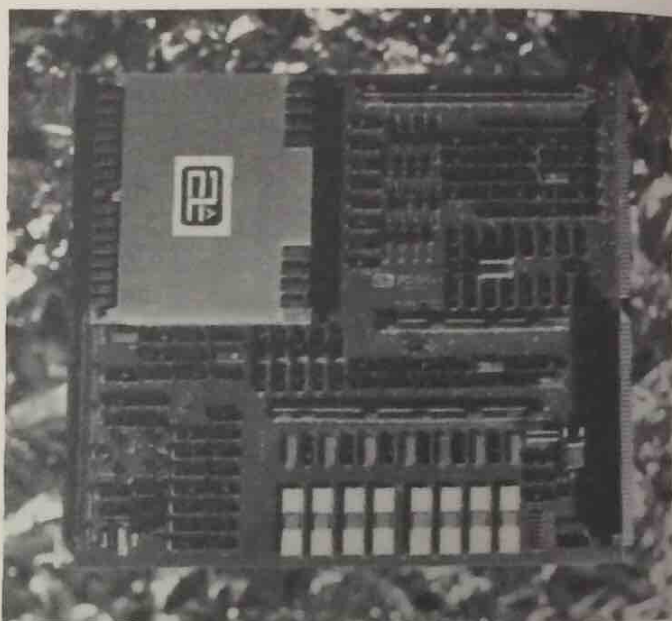
Preliminary data for the Toshiba chip indicate add and subtract times of about 10 μ s. And multiply and divide functions reportedly can be executed in 30 to 60 μ s. The chip operates from ± 5 -V supplies and dissipates 0.8 W.

Intersil's CMOS microprocessor benefits from the sizable software support that exists for the PDP-8. And the many designers familiar with the mini should be able to transfer that experience to the CMOS processor. However the unit's repertoire of eight basic memory-reference

instructions tends to limit the range of applications.

Still a multitude of applications uses the PDP-8. Noting this fact Intersil plans to develop a full set of circuitry and memory, all using CMOS, to operate with the processor. Conceivably, the end result could be a pocket-sized, portable PDP-8.

RCA, the leading CMOS manufacturer, also has a CMOS microprocessor. Called COSMAC, it is an 8-bit, two-chip set that emphasizes low-power dissipation and I/O interface capabilities in a proprietary design. Each chip dissipates only 100 μ W. And the I/O can be controlled



A complete mini, except for power supply, comes on this PC board. Computer Automation's Naked Mini/LSI-1 uses four custom LSI chips, each a 4-bit slice, and programmable logic arrays—rather than ROMs—to form the mini's central processing unit.

with up to 16 commands and a total of 23 lines, including a bidirectional data bus.

COSMAC is built around a 16 \times 16-bit scratchpad register, from which references to memory are made. The microprocessor can address up to 65-k bytes of memory, and it uses a simple two-step fetch-and-execute sequence.

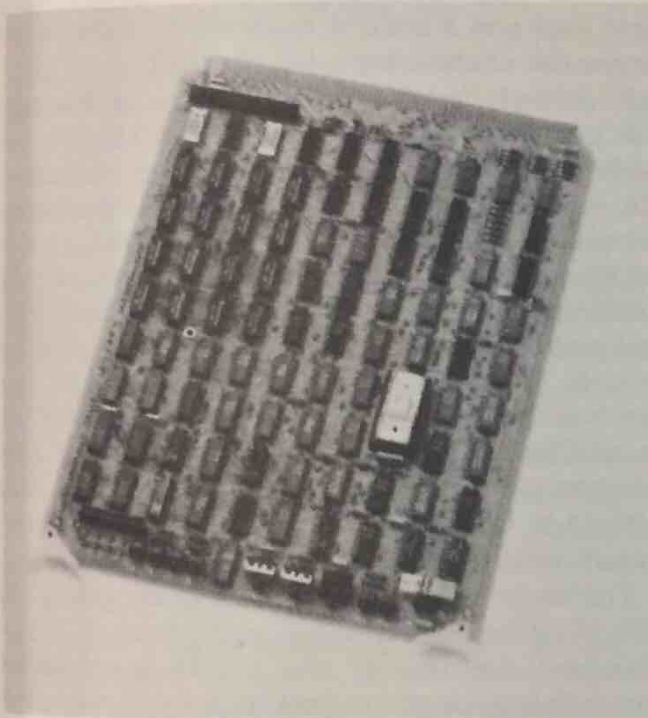
The processor cycle, consisting of eight clock pulses, ranges from 3 to 10 μ s. With the faster speed, and using a RAM with a 1- μ s cycle, a 6- μ s fetch-and-execute time can be obtained for any instruction. Response to interrupts, allowed only after complete instruction cycles, ranges from 3 to 9 μ s.

In the wings, RCA has a single-chip COSMAC. In addition the company is developing silicon-on-sapphire processors, with multichip units for military applications initially scheduled.

Micro-based Minis

Not to be undone by the emergence of component manufacturers into the microcomputer area, minicomputer makers have also embraced LSI microprocessors. Generally using custom MOS chips, they offer models aimed at high-volume applications—as are microcomputers—with large-quantity unit prices hovering about the \$500 level. This undercuts the price of many micros.

Micro-based minis are promoted as low-end



Custom silicon-on-sapphire LSI processors are the basis for General Automation's LSI 12/16, an 8-bit system. The mini maker also uses SOS chips for a 16-bit system, the LSI 16, that is a functional equivalent of the company's SPC-16 mini.

models complementing an established line. The new units benefit from the impressive software support—including extensive application programs—available with the rest of the line. Moreover the units offer traditional mini hardware/software features, which results in unique and versatile capability. At present Computer Automation, General Automation and DEC are incorporating microprocessors into their newer models. However, most industry observers expect that list to grow.

Computer Automation calls its microprocessor-based minicomputer the Naked Mini/LSI-1—an allusion to the fact that the mini comes complete, except for power supply, on a 15 × 17-in. PC board. The LSI-1 uses space-saving MOS

programmable logic arrays, rather than ROMs, to contain the unit's control logic. Four more MOS/LSI chips, each a 4-bit slice, make up the rest of the central processing unit for the 16-bit machine.

Standard features include direct-memory addressing and four other I/O systems. The unit also has hardware multiply and divide, multi-level indirect addressing, and up to 256 vectored priority interrupts. Memory can range from 1-k to 262-k 16-bit words, and may consist of core or semiconductor types in any combination. The basic processor cycle time is 1.6 μ s; an add-memory-to-register operation, involving 16 bits, takes 9.2 μ s.

A special feature of the machine's architecture is its ability to organize memory automatically. If memory modules of different size and type are mixed or rearranged on the bus, the computer assigns addresses without any reference to software.

The processor board contains slots for optional hardware functions. These include power-fail/restart, teletypewriter/CRT interface and real-time clock. In addition various plug-in options are available for additional buffers, drivers, interfaces and expanded memory.

Moreover the LSI-1 has an impressive 168 basic instructions. And most of these require only one memory location in contrast with the two or more locations usually needed in other minis.

The instruction set contains 29 memory-reference instructions—which simplify operations involving many pieces of data stored in memory. And conditional jumps can be performed with 63 instructions, each of which causes both the test and the jump. Other features of the instruction set include memory scan, three-way compare, word and byte addressing and full shift capability.

Among the various design aids available, Computer Automation offers several advanced software packages. These include Advanced BASIC, which can run with only 4 k of memory, Extended BASIC, using 8 k of memory, and FORTRAN. Standard packages include assemblers, loaders and debug and utility programs.

Mini turns to SOS

LSI processors employing silicon-on-sapphire techniques are used by General Automation in an 8-bit microcomputer—the LSI-12/16—and an expanded 16-bit model—the LSI 16. The SOS chips, developed for General Automation by Rockwell International, permit operation at speeds comparable with more conventional minis that use bipolar circuits.

The 8-bit unit, featuring 1 to 32-k words of

semiconductor memory and an instruction execution cycle of $2.64 \mu\text{s}$, comes with a low price tag of \$495 in 1000-unit quantities. It can be obtained on a $7\frac{3}{4} \times 10$ -in. PC board and in an enclosed system that contains power supply, battery backup for volatile memory and card slots for additional I/O boards.

For designers who want to retrofit new instructions in a ROM, the 8-bit LSI-12/16 provides a ROM "patch" system in addition to main memory. The patch eliminates the need to change ROMs when similar, but not identical, ROM programs are required. In addition the architecture of the LSI-12/16 has a shared-byte feature, allowing a two-byte instruction to be stored in a



A building-block approach to microcomputer designs is offered by Digital Equipment Corp. The modules, using Intel's 8008 PMOS processor, contain the peripheral circuitry usually needed to operate the microprocessor.

single-byte of memory. General Automation estimates that this architectural feature could reduce the amount of program memory otherwise required by about a third.

The LSI-12/16 uses 12-bit parallel addressing, which permits direct addressing of 4-k words of memory without paging techniques. It also provides eight 12-bit hardware registers, 52 basic commands, a processor-controlled priority-interrupt system and a teletypewriter interface. Standard control features include a relative time clock, external priority interrupt, 16-bit parallel I/O bus, integral console and a ROM-based console program. Several fail-safe features also protect against power transients and interruptions, component failures and program errors.

Software development can be minimized

through the use of a complete cross-program generation system on the company's SPC-16 minicomputer. The disc-based system provides assembly, editing and debugging. Other aids offered include a device-independent, real-time executive program and a conversational assembly system.

While the LSI-12/16 microcomputer can replace the company's somewhat slower SPC-12 minicomputer, General Automation's 16-bit unit is billed as the functional equivalent of the company's slightly faster SPC-16 mini. Furthermore the LSI-16 and expanded memory offers on two small boards the performance of the SPC-16 on six larger boards. The basic LSI-16 unit comes on a $7\frac{3}{4} \times 11$ -in. PC board, containing 1-k words of memory and an SOS microprocessor chip set. The two chips consist of an arithmetic logic unit and a control read-only memory, which stores the control logic for the ALU.

A second "micromemory" board of the same size provides a 32-k-word \times 18-bit (16 bits plus parity) memory system. The high density is achieved through the use of hybrid packaging. General Automation mounts eight 1103A (1-k bit RAM) memory chips onto a common ceramic substrate and plugs the substrates into vertical connectors on the board. A conventional approach, involving individual DIPs, would have resulted in a maximum board density of 16-k bits. In addition it will be possible to obtain a 120-k micromemory by replacement of the 1-k bit chips with 4-k bit RAMs, when they later become widely available.

Together with the micromemory board, the LSI-16 offers an average cycle time of $1.8 \mu\text{s}$. Standard features of the LSI-16 board include power-fail/restart capability, real-time clock, operation-monitor alarm, cold-start capability and an asynchronous memory interface. An additional PC board for options can be obtained with parity and hardware memory protection, teletypewriter controller, operator's console and a piggyback read-only memory board that can accommodate up to 4-k 16-bit words of memory.

MPS modules ease interfacing

A microprocessor module set is the initial entry into the microcomputer field by Digital Equipment Corp. The mini maker expects the modules, which use Intel's 8-bit PMOS processor, to be used as building blocks for dedicated controllers. Called MPS, the series can perform decision-making functions, add intelligence to data terminals and replace hard-wired logic systems. Moreover the units come complete with much of the peripheral circuitry usually needed to operate the microprocessor; hence interface

problems are reduced.

Five modules constitute the series. The CPU module contains the processor and complete instruction decode and control circuitry. Other modules provide 1-k to 4-k words of read/write memory and 256 to 4-k words of programmable read-only memory. A detection module can be used to obtain priority interrupts, while a monitor/control-panel unit can serve as a diagnostic checkout.

Software aids needed to program the MPS come in paper-tape format. The aids, offered in a special kit, consist of editors, assemblers, pROM programmer, debug software, duplicator and loader. Programs are prepared in conjunction with a PDP-8 with 4-k words of memory.

TTL/MSI processors are offered, too

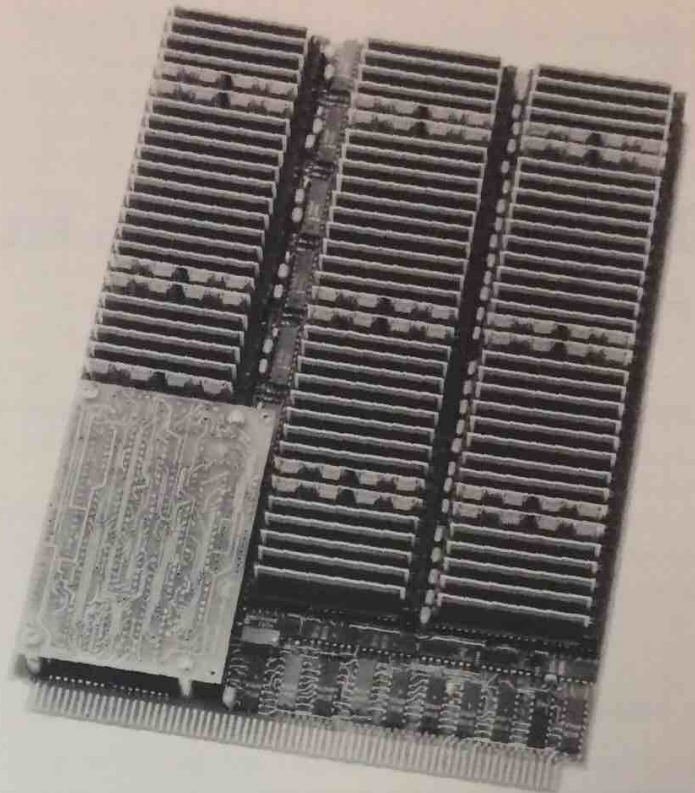
Still other microprocessor products use medium-scale-integration (MSI) standard-TTL circuits. These TTL/MSI processors, like their LSI cousins, can be programmed for a host of high-volume applications.

Complete TTL/MSI processors come on PC boards or in compact modules, and they sell for about \$1000 in single-unit quantities. They operate at faster speeds than their MOS counterparts, and they are offered by established vendors in the minicomputer business.

For example, 12-bit TTL/MSI units have been introduced by DEC, Fabri-Tek and Microdata. The DEC model, called the PDP-8/A Miniprocessor, is a two-module, MSI version of the company's widely used PDP-8 mini.

Fabri-Tek's MP12 processor includes 4-k words of core memory in a $2 \times 15 \times 9.5$ -in. module, and it requires only one 5-V, 40-W supply. And Microdata's Micro-One unit, on a single $8\frac{1}{2} \times 11$ -in. board, can be microprogrammed to emulate general or special-purpose computers.

The TTL/MSI processors use circuits that are widely alternate-sourced, in contrast with LSI units, which are virtually sole-sourced. But that picture could change dramatically in the next 12 months. Several manufacturers—Advanced Micro Devices, for example—are reportedly on the verge of following Microsystems International's lead in alternate-sourcing Intel's 8008 processor, or possibly producing Intel's 8080 chip. Aiming at more recent models, American Microsystems has plans to alternate-source Motorola's microprocessor chips. Furthermore a number of primary processor-chip suppliers are seeking other sources for their proprietary circuits.



Hybrid packaging yields a 32-k \times 18-bit 'micromemory' on a single board. General Automation offers the unit for use with the company's LSI-16 system.

Also expected are more bipolar/LSI microprocessors. For example, Texas Instruments—though tight-lipped officially—is said to be developing a bipolar unit that uses integrated-injection-logic (I²L) for increased density. I²L maintains the high speeds of bipolar circuits while achieving densities comparable with MOS (see "Integrated Injection Logic Shaping Up as Strong Bipolar Challenge to MOS," ED 6, March 15, 1974, p. 28). Most observers consider a high-density process, such as I²L, absolutely essential to achieve the high functions per chip needed for bipolar/LSI.

The strong emergence of bipolar/LSI microprocessors could have far-reaching effects. At present processor-chip vendors, who are the hardware experts, are increasing their software support, while mini makers, who excel in software, are stripping down to "bare-bones" models.

However, direct competition of products generally is limited because of the speed gap between most micros, which use MOS, and bipolar mini models. Bipolar/LSI processors could bridge the speed gap, thereby sparking a historic confrontation between chip vendors and mini makers.

Microprocessors in Test Equipment

Promises and Delays

STANLEY RUNYON

Associate Editor, Electronic Design

Though powerful, the microprocessor hasn't quite made it yet in test and measuring instruments. Most microprocessors are currently finding homes in commercial, OEM and computer-oriented products—such as traffic-control systems, blood analyzers, data-entry terminals, point-of-sale terminals, check processors and automatic typesetters.

Exactly what is it that has kept the microprocessor—a glamour product if there ever was one—from being more prolifically used in test instruments? Several reasons.

For one, it's relatively expensive. While the CPU by itself may cost under \$100, the hardware cost can zoom when all the needed memory, I/O devices, clocks and other ICs are added.

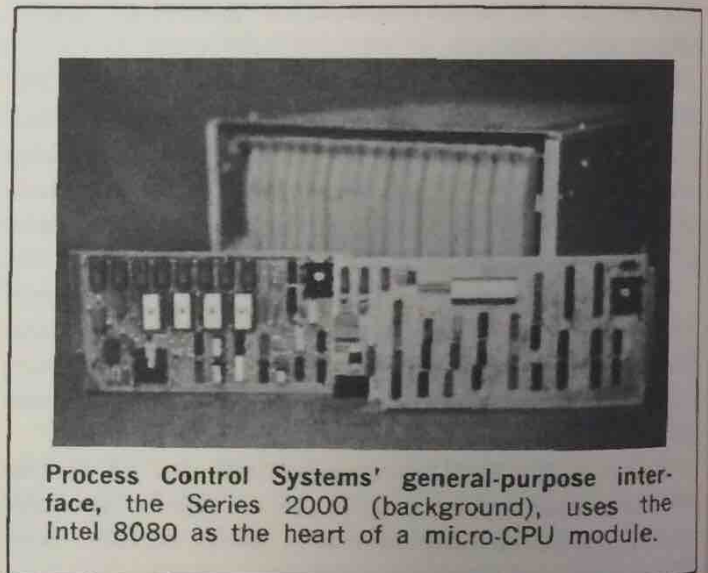
For another, the microprocessor's relatively sluggish data cycle limits its use in all but low and medium-speed applications.

Add to these limitations the minimal software support now on hand, then top it off with a lack of familiarization among designers used to hardwired logic. (And don't forget that, up to now, only two vendors offered them.)

Rethinking design habits

All of this points to a time delay, probably two to three years, before the microprocessor surfaces as a major circuit element—a time delay during which prices must tumble, performance must rise and engineers must learn to design with software rather than hardware.

It's true, however, that practically every major instrument manufacturer has bought one. But they've been purchased only for evaluation and not for a specific instrument. Thus, at least for most instrument vendors, the microprocessor is a



Process Control Systems' general-purpose interface, the Series 2000 (background), uses the Intel 8080 as the heart of a micro-CPU module.

solution looking for a problem.

Right now, probably the only measuring instrument with an internal commercial microprocessor is Boonton's Model 76A Capacitance Bridge.

In this automatic, programmable unit, an Intel MCS-4 microprocessor controls the various front-panel, display and I/O functions, as well as digitally controlling the bridge balance and computing the unknown capacitance, conductance, Q factor and dissipation (D).

On top of this, the 76A automatically corrects all predictable bridge errors, autoranges over 0 to 2000 pF and digitally displays the results.

Boonton's solitude isn't to suggest that the microprocessor—that is, a general-purpose MOS or bipolar LSI CPU on a chip, surrounded by ROMs, RAMs, I/O devices or other ICs—isn't being applied. It is. But the viable products merge into a trickle rather than a torrent.

And even though it's a sure bet that the next few years will bring microcontrolled DVMs, synthesizers, sweepers and other instruments, it's not so sure that the microcontroller or micro-computer role will be played by a commercial microprocessor.

One reason for this is that some potential customers, big companies like Hewlett-Packard, have opted to design their own hardware or firmware to implement the concept of a microprogrammable processor. Or they are building what can be termed a specialized microprocessor.

And, of course, the role of a built-in digital controller, calculator, processor or data shuffler can be, and is being, filled by ROMs, RAMs, PLAs and random logic—all rivals to the commercial microprocessor.

Examples are HP's 3330 frequency synthesizer, which has an internal digital processor, and the Tektronix' 7704A Digital Processing Oscilloscope, a unit that can perform calculations on its input signals.

And Computer Automation, a well-known mini-computer vendor, offers a one-card mini with a rather complex MOS/LSI PLA (which some people mistakenly label a microprocessor).

Because of this functional rivalry, the definition of a microprocessor is often confused. There's little doubt that, more and more, the word microprocessor will come to mean the MOS/LSI—or bipolar—component and not the microprogrammable processors currently being implemented in MSI, SSI, and with minicomputers or even large-scale computers.

It should be noted that some commercial microprocessors—such as National Semiconductor's GPC/P and the AMI 7300 (from American

Microsystems, Inc.)—are microprogrammable.

Illustrative of the microprocessor concept is the Model 400 graphic-display system built by Adage Inc. of Boston. Because microprocessors of the commercial variety couldn't handle the fast I/O required by the refreshed CRT graphics, Adage built its own microprogrammable display processor.

To get the speed, Adage couples fast Schottky TTL MSI with either a high-speed bipolar RAM or a fusible-link pROM, depending on the configuration. The result: a processor with a multiplication time of 240 ns.

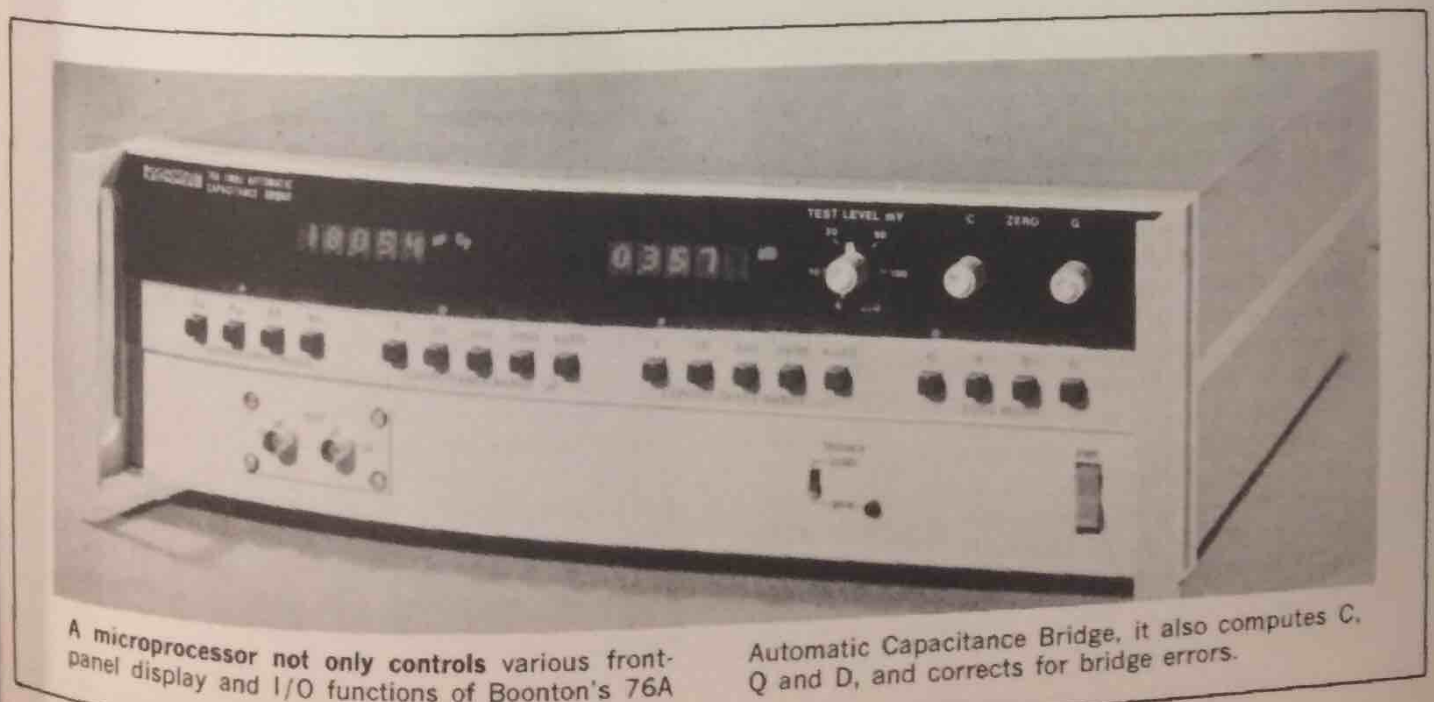
In contrast, Iomec, Inc.—a Santa Clara-based firm—was able to use the Intel MCS-4 in its Porta-verter line of remote data-entry terminals. The big difference, of course, is that data entry is a relatively slow job.

Terminals: A natural home for microprocessors

For Iomec, the commercial microprocessor appears to have been a godsend. When it became apparent that the Porta-verter's original LSI hard-wired logic chips weren't going to be delivered on time for Iomec to meet its delivery commitments, the company was able to replace the chips with off-the-shelf microprocessors at about the same cost.

And not only did the microprocessor fulfill the terminal's original design goal; it allowed Iomec to offer such extras as special character keys, operator prompts, error correction and unattended operation. Even with all options, Iomec says its price is two to eight times less than that of intelligent terminals.

One intelligent terminal that was conceived



A microprocessor not only controls various front-panel display and I/O functions of Boonton's 76A

Automatic Capacitance Bridge, it also computes C, Q and D, and corrects for bridge errors.

from scratch with a microprocessor as the central control unit is the Microterm series from Digi-Log Systems, Horsham, Pa.

Because of the microprocessor and the Microterm's modular design, Digi-Log reports, designers with unique terminal requirements can just about write their own specs. Such features as display formats, code structure, baud rate, keyboard format and editing can be personalized if custom-written software is combined with selected hardware modules.

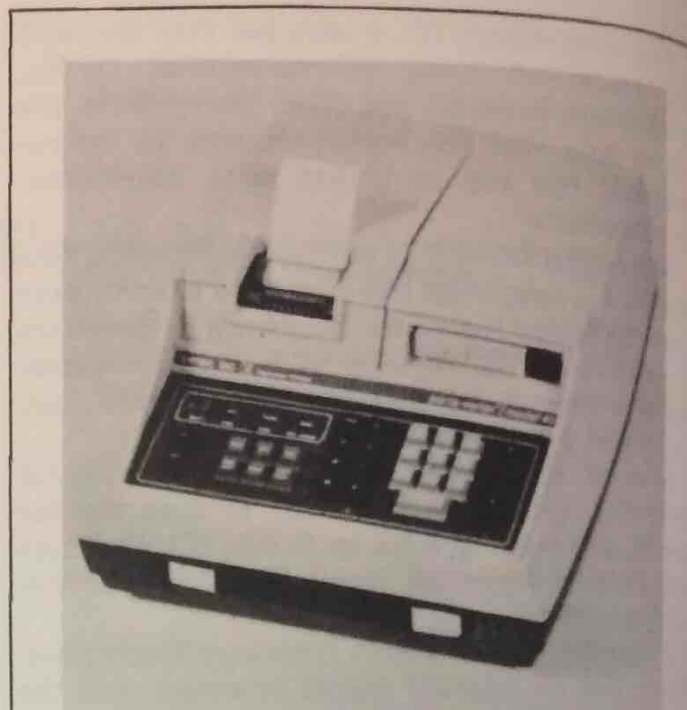
To get the identical capabilities with a hardware design, Digi-Log estimates that 600 to 700 ICs would have been needed.

Applications such as Iomec's and Digi-Log's—in which the microprocessor replaces hard-wired logic or minicomputers—now dominate the product scene and are likely to do so for the next few years.

In a blood analyzer built by Helena Laboratories of Beaumont, Tex., for example, a microprocessor bumped hardware to provide more complex, faster and more accurate data handling—in a smaller package yet at less cost.

The analyzer, through the microprocessor, performs mathematical and operational routines on raw data received from a sensor and gives a printout, in medical units, of blood protein content.

In still another application, four PC cards were knocked out by Intel's 8008 microprocessor in a computing integrator built by the Autolab Div. of Spectra-Physics.



This remote data-entry terminal—the Porta-verter from Iomec, Inc.—offers intelligent operation. Credit goes to a four-bit microprocessor.

Designed expressly for chromatographic data processing, Autolab's System 1 can replace a general-purpose mini or a programmable calculator. Autolab notes these benefits with the microprocessor: The design was simplified, the instrument shrank, reliability went up and costs were reduced.

How did Autolab and Helena save money by

A 'micro' what?

What's a microprocessor? Confusion seems to result when the prefix "micro" is plugged into the front end of a word. And no wonder. Generic names are few, and vendors like to coin words to describe their "unique" products. Here is an attempt to clear the confusion:

Microprocessor: An LSI central processing unit (CPU) on one—or a few—MOS or bipolar chips. Along with arithmetic functions, the CPU may perform input/output (I/O) jobs and may contain a scratch pad or other memory. To form a working system, at least one external ROM, RAM or other memory device is usually used with the CPU.

Microcomputer: Microprocessors are sometimes called microcomputers. But so are "small" computers—such as bare-bones minicomputers, built on one or a few PC cards. Minicomputers built around a commercial microprocessor are often called microcomputers.

To make things worse, computers that use

microprogramming are sometimes referred to as microcomputers, regardless of the size and packaging of the CPU.

Microprogrammed processor or computer: The term "microprogrammed" refers to any computer whose instruction set is not fixed but can be tailored to individual needs by the programming of ROMs or other memory devices. Consequently whether the computer is a mini, midi, maxi—or a microprocessor—theoretically it can be microprogrammed.

Microcontroller: Another all-purpose word, this can mean a microprogrammed machine, a microprocessor or a microcomputer used in a control operation—that is, to direct or make changes in a process or operation. But there's a narrower definition, more in keeping with the prefix, in which microcontroller refers to any device or instrument that controls a process with high resolution, usually over a narrow region.

substituting microprocessors, while others point to their high cost?

The answer, of course, depends on what the microprocessor replaces. For simple jobs with few components, random logic is less expensive. But at the other extreme, if the microprocessor shoves a minicomputer aside, the cost savings—as well as the volume reduction—can be substantial.

Micros vs minis and other hardware

Dr. Robert E. Jackson of Applied Computing Technology, Irvine, Calif., offers this rule of thumb to determine the cost tradeoff between random logic and microprocessors: A 50-IC system costs about the same as a microprocessor plus 20 interface ICs. Included in the rule are parts cost and the cost of handling, testing and interconnections.

Does the microprocessor's inroads into minicomputer territory worry minicomputer vendors? Not yet. Punch for punch, the microprocessor—at least at present—is no match for the mini in speed, word length, memory capacity, number of instructions, computing power and software availability.

But where the mini is being used as a 50-mm cannon when a BB gun will do, watch out. In dedicated control applications that don't need complicated calculations, the microprocessor can fill the need for a programmable device; a mini here would be overkill. Where a mini is used



The Microterm series, built by Digi-Log Systems, are microprocessor-based, intelligent terminals that can be customized to a user's specs.

primarily to compute but a built-in computer with low power consumption is preferred, a microprocessor may slip in.

Road traffic controllers, such as the Model 901 from Multisonics of San Ramon, Calif., are applications in which microprocessors may also replace minis. The job is so easy for a mini that it loafs half of the time. Buried in the 901 is Intel's 8008 microprocessor. This 8-bit unit—coupled with 14 pROM pages and other memory—forms the heart of a computer that, though invisible to the user, optimizes traffic flow in many separate intersections.

The advantages of the microprocessor approach here? Foremost, the manufacturer can tailor the system to individual applications by first changing the software and then just plugging in new ROMs. Other benefits include a significant size reduction and easier troubleshooting and maintenance.

Micros turn up in minis

Apparently the minicomputer makers aren't too concerned about a potentially serious microprocessor threat. In fact, one mini vendor—General Automation—has just unwrapped a 12-bit microcomputer that uses a silicon-on-sapphire (SOS) microprocessor made by Rockwell International.

General Automation defines a microcomputer as a full system computer on a board, with the processor on a single chip.

Digital Equipment Corp.—the largest mini manufacturer—also appears to view the micro-



Microprocessors bring sophistication to video games, such as Atari's Quadrapong—a hockey-like diversion for four people.



By replacing a minicomputer with a four-bit microprocessor, Data Type Corp. was able to cut costs

and slim down the size of its newspaper-oriented optical page reader—the DFR 300.

processor as a friend and not a foe.

DEC has unveiled a dedicated control system built around a microprocessor—the MPS Series. Consisting of five modules, the MPS can be built into terminals, process-control systems and other instrumentation.

Thus we can anticipate a three-way rivalry for equipment space—with microcomputers getting squeezed from below by microprocessors and from above by bare-bones minicomputers.

However, some engineers feel that just as the introduction of the mini spawned entirely new applications, rather than cut into the applications of medium and large-scale machines, so will microprocessors and microcomputers.

But what will happen as microprocessors grow in complexity, speed and power, and the line between micro and at least the low-end mini begins to blur? Perhaps we'll see a vertical shift, with minis creeping into applications now dominated by the medium and big computers.

In the meantime newly conceived products—like the Staid Corp.'s Datacash, a point-of-sale terminal—will use microprocessors instead of hard-wired logic. And many existing products will no doubt convert to microprocessor-based designs to compete.

In Staid's case, an MCS-4—coupled with pROMs—allowed the company to build a terminal that is easily tailored to a customer's needs. Datacash—primarily intended for restaurants and cafeterias—has 110 keys, each of whose legends can be changed on site with a peg matrix, plus six function keys for special applications. And the unit keeps a running inventory of each key strike. With hard-wired circuitry, Staid says, 30 TTL cards would have been required.

One company that has already converted an existing product—at the expense of a mini—is Data Type of Miami. The company's DFR 300

optical page reader started out with either PDP-16s or NOVA 1200s as the code-translator controller.

But, says Data Type, the PDP-16 costs substantially more than the MCS-4 that replaced it, and the NOVA 1200 was three-quarters asleep on its job as a start-stop controller, error detector and buffer.

At present the MCS-4 controller portion of the page reader, which reads font for the newspaper trade and functions as a terminal in the graphic-arts, has been designed as a separate, retrofit package to replace the mini directly. Future models, however, will have the microprocessor built in.

Distributed intelligence—a new trend

Some applications enable the microprocessor and the mini to work together symbiotically. This appears to be the case with the Telecontroller—a programmable, front-end data-communications processor built by Action Communications of Dallas, Tex.

Instead of replacing a mini, an Intel microprocessor in the Telecontroller freed the computer to take on more work. The microprocessor, which serves as a message switcher in a high-speed binary interface, efficiently handles the high-speed data and then passes it on to the mini.

With an intelligent function distributed in this way, more computer core becomes available, and many more terminals can come on line to be served by the mini. Thus look for distributed intelligence—via microprocessors—to appear in more and more new terminal systems and to be retrofitted to existing systems.

One terminal supplier that will probably be doing both is the Digital Systems Div. of Texas

Instruments. TI sees the microprocessor as an opportunity to give its customers a "rubber" terminal system—one that can grow as data-processing functions change.

Intelligence was redistributed in still another interface—the Series 2000, by Process Control Systems. In this general-purpose process-control interface, a micro CPU and memory were designed to slip in and, in some applications, replace a mini. Built around the 8-bit Intel 8080—the first n-channel MOS microprocessor—the interface can be used as a down-line satellite of another computer or as a stand-alone data-acquisition system. In the latter case the interface bumps the computer.

By building in the microprocessor, Process Control Systems not only saved hardware dollars, but cabling costs in remote processes were also considerably reduced. For example, Ford Motor Co. uses a number of the interfaces at carburetor-flow tests stands. Here the micro CPU serves as a satellite to a down-line Interdata 70, and it multiplexes the interface data to the computer. In a similar application, the Buick Div. of General Motors puts the interface at its assembly line in a torque-monitoring system.

In the torque monitor, the micro CPU converts parallel data to serial and funnels the information stream into a dual-conductor cable. The cable then carries the data downstream, where it is finally dumped into an HP 2100 mini. Another interface, between the computer and the cable, reconverts the serial stream to parallel data.

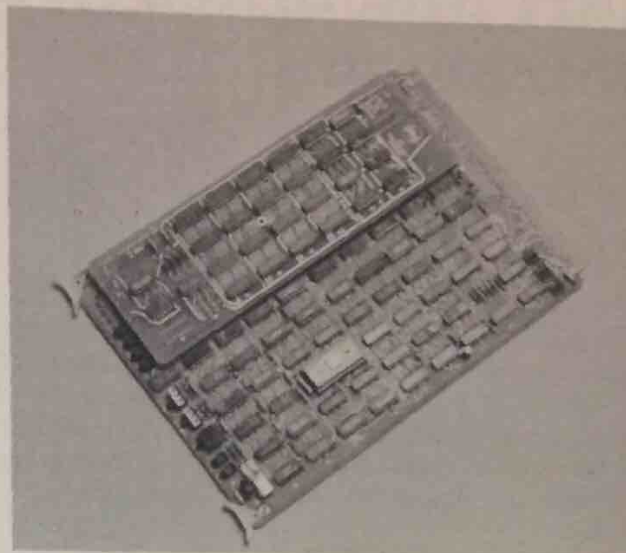
Other microprocessors surface

The 8080 microprocessor used by Process Control Systems in its interface is Intel's newest and most powerful unit and is sure to boost the semiconductor company's competitive lead time even more. Because of the lead, Intel now dominates the market in terms of available microprocessor-based products.

But National's IMP-16C microprocessor and GPC/P MOS/LSI system kit are being built into prototypes and may soon pop up in such applications as aircraft control systems, flight-status indicators, analog computers (strangely enough), nuclear instrumentation and symbol generators.

And, of course, the other semi houses aren't sleeping—AMI, General Instrument, Electronic Arrays, Motorola, Signetics and Western Digital will all have general-purpose units available soon. RCA has already announced the first CMOS microprocessor, which it expects to offer initially on a sampling basis.

With all this activity in microprocessors, it's only natural for a designer to ask: Is there one



General Automation's LSI-12/16—an 8-bit automation microcomputer—is built around the first silicon-on-sapphire microprocessor.



Staid Corporation's Datacash—a microprocessor-based point-of-sale terminal for restaurants—keeps an inventory of every key strike.

in my future? The answer: Inevitably, yes. In the meantime, though, designers are more likely to run into one at their ophthalmologist's, at the butcher's or at their favorite watering hole.

At the eye doctor, look for Coherent Radiation's Dioptron—an optical instrument that measures and analyzes the eye's focal characteristics and prints out the results.

At the meat market, check for the butcher's thumb on a Toledo digital-computer scale—a unit that converts weights to prices, prints labels and displays unit price, weight and total price.

And at the bar, try to beat the gal standing next to you at a new video game, from Atari, in which a cat chases a mouse through a constantly changing maze. If you lose, the game—but win the lass—thank the microprocessor.

Smart Machines in Industrial Electronics

JOHN F. MASON

Associate Editor, Electronic Design

In the midst of a recession and the normally fluctuating cyclical demands, industrial electronics is steadily undergoing radical changes in design—due, to a great extent, to the growing acceptance of the microprocessor.

More machines are being built "smart"—with computational capability provided by microprocessors. And the cost for such performance is one tenth or potentially one fiftieth that of a minicomputer.

The sectors of the electronics industry that will suffer from this invasion of the tiny, cheap, intelligent chips include the manufacturers of conventional TTL circuits and the small signal resistors and capacitors that are used with these circuits on PC boards. Except for high-speed applications, these components just won't be needed.

But emerging from the financially painful demise of these and other components there is good news for the design engineer. With microprocessor chips he is finding that he can produce a much better product more quickly than before, at a lower cost. He has more design flexibility and much shorter design cycle times.

Equipment benefiting from the microprocessor chip sets include instrumentation, communications, process-control systems, industrial machine tools and a vast number of small "dedicated" tools—tools that can perform one specialized function.

But the invasion of microprocessors is far from complete. General Electric, an established stronghold for the manufacture of numerical control (NC) equipment, has introduced its newest softwired numerical control system, the Mark Century 1050 series, which uses microprocessors. Using advanced diagnostic techniques, the 1050 is built for use in ma-

chining centers that use high speed machines. But according to the company's James Connolly, product manager for numerical control in Waynesboro, VA, hard wire numerical control will be around for a long time.

Westinghouse is looking toward microprocessors, too, but says: "Minicomputers are coming down in price and will continue to be used for some time."

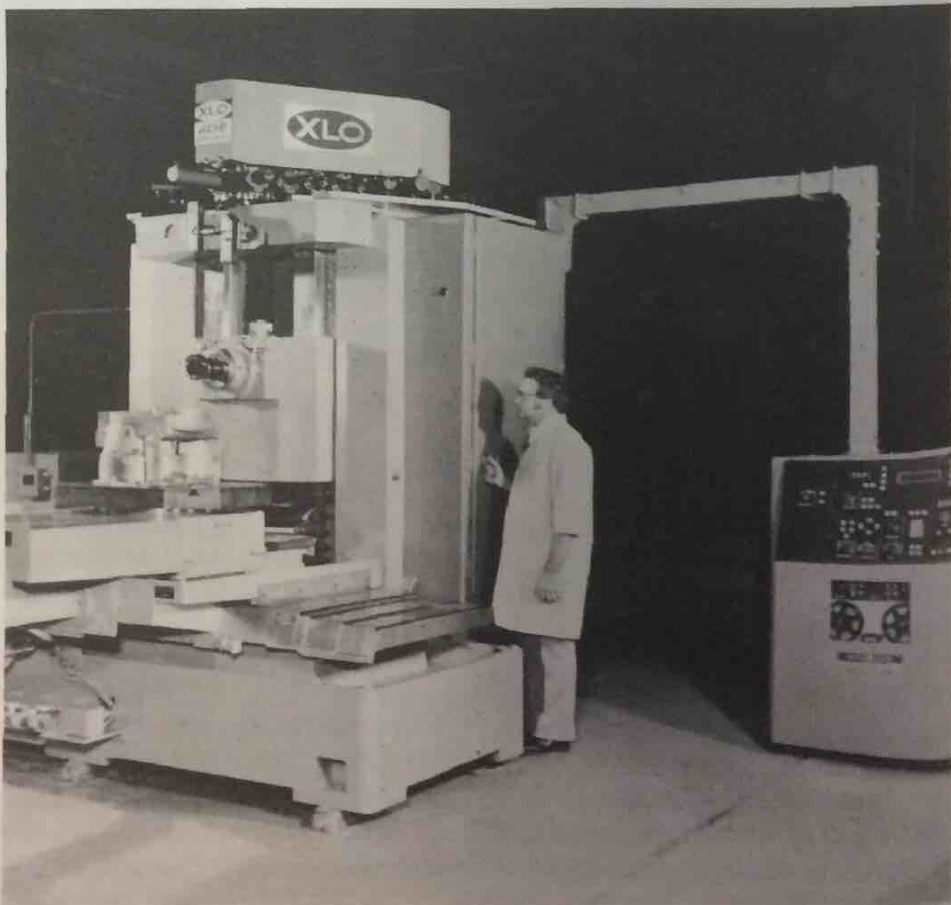
Robert Morgan, marketing manager for the company's industrial equipment division in Buffalo, NY, sees that technology in 1975 will evolve slowly without many surprises. He says: "We're continuing to move more into solid state—thyristors—and diodes—a change

that has been slightly slower than we'd expected."

Because of the business climate and the environmentalists, Morgan continues, "We're going to start making use of a lot of technological developments we've already got—particularly in the area of energy conservation."

Morgan continues: "Rather than innovation, we're going to take a hard look at the way our equipment is built and the power sources it uses. We'll take SCRs, for example, and convert them a bit and apply them in different ways."

"Conditions in 1975 are going to force a lot of technology that's been around a long time to be used in quite a number of new



General Electric's 1050 softwired numerical controller—which employs the latest in microprocessors—operates a multi-axis machine-tool work center.

ways," Morgan concludes.

Allen-Bradley looks forward to a bumper year for computerized numerical control (CNC). "We began to get a foothold in this market in 1973," says Mike Gregory, product manager for numerical control equipment, Cleveland, OH. "It expanded in 1974 and will grow even more in '75."

People are familiar with the equipment now, Gregory explains, and they no longer fear abandoning hardwire numerical control for a more "intelligent" system. And also the cost of CNC is coming down. "The price of CNC is really approaching that of hardwire NC," Gregory says. "To give a hardwire system the flexibility that CNC has would cost far too much."

Allen-Bradley continues to build Direct Numerical Control (DNC) systems but does not expect to see growth in that particular sector this year.

Gregory reports: "At the simpler end of the spectrum, programmable controllers, which began to catch on in 1972, are expected in 1975 to outstrip their tremendous growth of the past year."

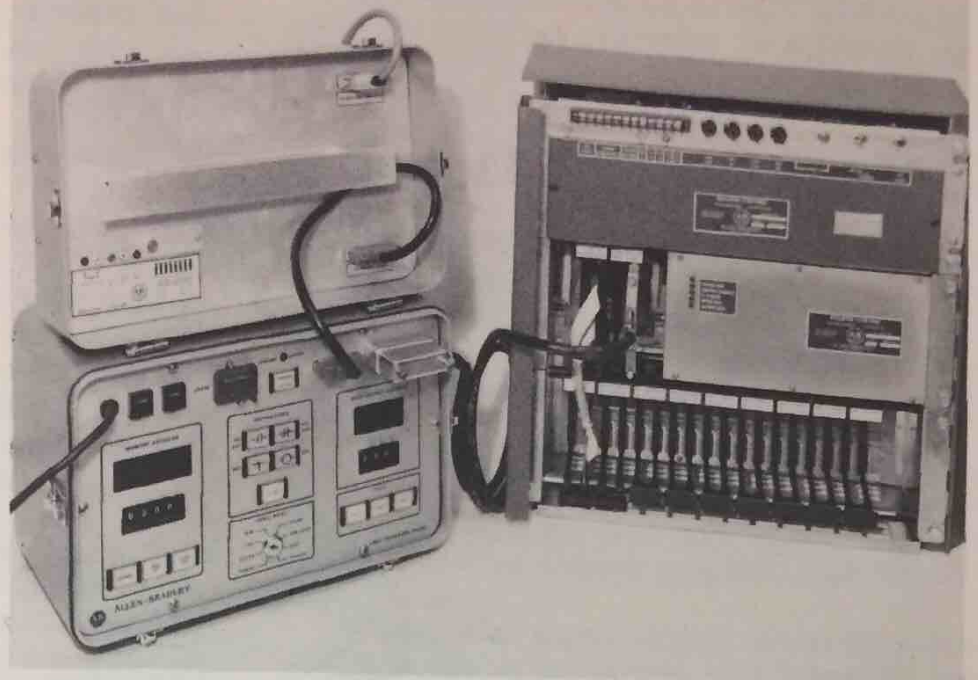
One drawing card of the programmable controller, of course, is its use of microprocessors and its sequencing and computing functions. The machines are becoming cheaper as well as more powerful.

Allen-Bradley believes the capital-goods market is going to remain strong throughout 1975. "The growth won't equal that of past years but we do expect sales in 1975 to be better than 1974—especially for CNC systems," Gregory says.

Cincinnati Milicron sees more use for microprocessors in 1975, particularly in programmable controllers that require a lot of relays, according to Charles F. Carter, the company's director of Product Development for the Machine Group.

Carter sees gains in '75 by both CNC, computerized numerical control and hardwire NC. CNC offers more than NC, and its cost is coming down. But on the other hand, some of the sophisticated features of CNC are being incorporated into NC.

Adaptive control—publicized at



Programmable controller from Allen-Bradley offers a convertible memory system. Either a programmable read-only memory processor or a read/write memory processor may be used interchangeably. Shown are the program panel (left) and basic controller system.

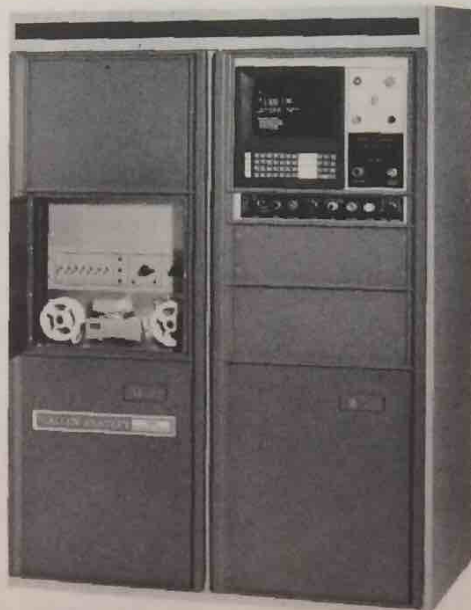
the last three Machine Tool Shows in Chicago—is still just getting off the ground. Its main acceptance thus far, Carter says, is by the aerospace industry.

As for new markets, U.S. companies do not seem to be looking toward the oil-producing countries as aggressively as the European electronics industry does. Many Americans queried feel that the

market is not big enough yet. They say the Arabs' needs at this time won't make up for the declining needs of such former big spenders as England and Japan. Europeans, on the other hand, seem willing to cultivate the Arabs and wait. Companies such as Bruel & Kjaer in Copenhagen, for example, say they are already looking at new designs for their equipment, to simplify it for operators who are not skilled.

The first big sales to the oil producers will probably be consumer products—black and white television sets. But Iran and Saudi Arabia plan to import entire industrial plants. This market won't burst wide open in 1975, but Europe is waiting to step in when it does.

As for the cyclical demands for industrial equipment—a concept respected by many: Spending is expected to continue through March, drop down for the second and third quarters and rise slightly in the fourth. Consumer buying will pick up in early 1976, bringing capital-goods spending up by the second half of '76. So the end of the dark tunnel we're in now will bring us out happy and prosperous right in the midst of the nation's bicentennial anniversary celebration—just as if the whole thing had been planned.



Computerized numerical control systems, such as the 7360 model from Allen-Bradley, can be readily shifted from one type of machine operation to another without hardwiring changes.

Micro vs Mini

Mini Manufacturers Counterpunch

JOHN F. MASON

Associate Editor, Electronic Design

First there was the giant computer. The data processor to end all data processors. Centralization was the answer. One big machine would do the brain work for an entire facility.

Then came the smaller machines. Decentralization became an acceptable concept. But was it really the way to go?

Then the minicomputer arrived. This was real revolution! Decentralization was right, after all. And the mini was king!

And then—and who was ready for this?—the microprocessor appeared.

Will there be a battle? Is it curtains for the mini? For Digital Equipment Corp. For Data General? For Hewlett-Packard?

Hewlett-Packard's Paul C. Ely Jr. says there will be a battle but the mini will survive—if the manufacturer knows what he's doing. Ely is general manager for the company's Computer Systems Group in Cupertino, CA.

Changes are taking place in the computer industry all right, Ely agrees. The microprocessor is having an impact.

"But fortunately," Ely explains, "the minicomputer has always enjoyed the advantage of having a very broad base, as far as applicability is concerned. As industry grows and makes new demands, the minicomputer moves in to fill them.

"The microprocessor and the minicomputer aren't at war but the microprocessor has captured some territory and is forcing the mini to move. The mini has consistently come down in price and up in performance. So the microprocessor is simply taking over the smaller, cheaper jobs that the mini is now too big to handle. Together, the microprocessor and the mini have expanded the market."

Ely sees three important trends in the giant minicomputer companies:

1. Manufacturers rarely sell someone a bare

CPU these days. They provide the mini in a complete system with integrated peripherals. "Even the OEMs want complete systems," he says. This trend, which actually started about three years ago—before microprocessors—is evident in many product lines: "DEC's PDP-11/70, Data General's Eclipse, and HP's 3000 CX," Ely notes.

2. A system's software is more powerful than it used to be. "You give them really comprehensive operating software that will solve their problems," Ely says. As a result, less money is spent for developing custom software and a better, more reliable software results.

3. Manufacturers have got to build their own peripherals. This gives the systems manufacturer a chance to make a unique performance contribution to his product at the systems level. It also gives him control of his reliability and costs.

How to stay alive

Some minicomputer houses will be around two years from now and some won't, Ely says, adding: "HP and apparently DEC are actually experiencing a period of growth during these generally bad economic times. To survive, you have to move forward into systems integration and peripherals. And you must also move backward into IC technology."

As an example, Ely notes: "In the past almost any group of engineers with a bright idea could buy logic from Fairchild or Texas Instruments or Motorola, build a CPU, put together a simple, though perhaps clever, software package to run it, and they were a 'mini supplier.'"

"These are the people who may be replaced by the microprocessor companies," Ely predicts.

To survive in the long run, he says, manufac-

turers must know how to build electromechanical devices and at least the main systems peripherals, such as discs and tapes, for mass storage and for line printers and terminals to interact with the system.

"This is where the money in a system is," Ely says, "not in the CPU.

"Apart from cost," he adds, "reputation is at stake. If you buy your discs, you won't be able to differentiate them from those of your competitors. And you can't claim that your discs are better unless, in fact, they are and unless you make them. As the industry matures, the ones who have developed their own capabilities in these key peripherals will be successful.

"This doesn't mean that we're going to stop buying all peripherals. But we do plan to build a large share of the ones that represent the essential performance limitations in our systems."

Expertise in LSI design is also required, Ely says. Minis of the future won't be built with standard circuits from a semiconductor house; they'll be custom-designed, he predicts.

"Some of the minicomputer companies are vertically integrating backwards into IC technology by working closely with a supplier," the HP manager points out. "That's what DEC has done with Western Digital to make their LSI-11. This is an LSI version of the DEC PDP-11. We in HP have invested in an in-house LSI capability and Data General has bought an LSI facility."

Ely feels strongly enough about the need for expertise in IC technology to predict: "The extent to which we are successful in this will determine our competitive success in the long run."

Will the IC houses become systems producers?

"Microprocessor companies don't at this point have the software capability to become a systems house," he says. "They can, however, acquire this by first recognizing the need for it, hiring a competent staff and gaining experience.

"Developing a peripheral capability, on the other hand, is more difficult. This requires both an engineering and a manufacturing capability—something that's very different from manufacturing ICs and that calls for a substantial investment. Making a good disc is a sophisticated electromechanical job which requires major machine shop facilities."

More uses are being found for the minicomputer because new industrial areas are discovering its applicability, and also because it's now being offered as a more complete, practical system, Ely says.

"OEMs, who began buying relatively simple CPUs to use as controllers, now frequently put a whole computer system in their product. They build sophisticated computer-controlled systems into products that formerly would have received no more than a simple disc-based unit.

"We see a lot of new business in the energy-related industries. We are selling computers to OEMs who sell to exploration companies, service companies and manufacturers of drilling equipment. There are a large number of small technically based companies that serve the energy field that are in a rapid state of growth today.

"A great deal of training and patience are required by the user of today's minicomputers," Ely says. "I personally think that HP has done a good job in this area, but we're going to do even better."

SECTION II

Getting Started: *Microprocessor Basics*

Following a discussion of hardware characteristics, three software-oriented articles deal with the capabilities of microprocessor instructions and different coding tools. An algorithm for a traffic-light controller illustrates principles that can be used with any microprocessor.

Next, a piece on input/output characteristics of competing microprocessors shows how the I/O affects peripheral-interface capabilities. The final three articles concentrate on initial design phases.

MOS/LSI Microprocessor Selection	33
<i>Alan J. Weissberger, Microprocessor Applications Engineer, National Semiconductor, Santa Clara</i>	
Analysis of Microprocessor Instruction Sets	37
<i>C. Dennis Weiss, Ph.D., Bell Telephone Laboratories, Holmdel</i>	
MOS/LSI Microcomputer Coding	45
<i>C. Dennis Weiss, Ph.D., Bell Telephone Laboratories, Holmdel</i>	
Traffic-Light Controller: An Example	51
<i>C. Dennis Weiss, Ph.D., Bell Telephone Laboratories, Holmdel</i>	
Microcomputer I/O Capabilities	56
<i>Andre G. Vacroux, Bell Telephone Laboratories, Holmdel</i>	
Microprocessor or Random Logic?	62
<i>Donald R. Lewis and W. Ralph Siena, Automata Systems Corp., Kew Gardens, N.Y.</i>	
The Anatomy of a Microprocessor Chip	67
<i>Donald R. Lewis and W. Ralph Siena, Automata Systems Corp., Kew Gardens, N.Y.</i>	
Clearing the Interface and Software Hurdles	73
<i>Donald R. Lewis and W. Ralph Siena, Automata Systems Corp., Kew Gardens, N.Y.</i>	

MOS/LSI Microprocessor Selection

ALAN J. WEISSBERGER

Microprocessor Applications Engineer
National Semiconductor, Santa Clara

With the suddenly popular MOS/LSI microprocessor turning up in many new and intriguing applications, some down-to-earth questions are confronting designers: Is a microprocessor right for my application? If it is, which one should I choose?

Microprocessors are used primarily to replace or upgrade random-logic designs.¹ Selection problems can be simplified by careful analysis of hardware requirements, software capabilities and the design aids offered by manufacturers.

A microprocessor is more efficient than a random-logic design when the following conditions are present:

- About 50 or more ICs are used in sequential or combinational designs to perform many functions.
- Functional flexibility or expansion capability is desired.
- Random collection and routing of data are required.
- Complex, logic decisions must be made. (Array logic—in the form of ROMs, pROMs and

PLAs, or programmable logic arrays—is another alternative for this condition.)

▪ Arithmetic computations are needed. (Bipolar 4-bit arithmetic logic units and some of the newer logic devices can also fulfill this need.)

▪ Speed requirements are not excessive. Although microprocessor speeds are not high, the use of wide-word, or multiple microprocessors, can increase throughput.

Microprocessors can be used in any quantity. For high-volume—more than 25,000 units—a small number of microprocessors can be used for prototyping or pre-production runs before you go to a custom LSI design.

A listing of some microprocessors and vendors appears in Table 1.

What are the key features?

The importance of individual microprocessor characteristics depends heavily on the application. Here is a checklist of key features to consider before making a selection.

Table 1. Some microprocessors announced and expected

4 bit Structured chips, cards, systems	4 bit Building block chips	8 bit Chips, cards, systems	12 bit Chips	16 bit Cards	Custom Chips
<ul style="list-style-type: none"> *Rockwell PPS-4, 20102D02 *Applied Computing Technology PPS-4MP, CBC-4, prototyping systems (for PPS-4 and MCS-4) *Intel 4004, MCS-4, Intellec 4 *Fairchild PPS-25 National Semiconductor IMP-4, FILU 	<ul style="list-style-type: none"> *National Semiconductor RALU, CROM Monolithic Memories bipolar LSI controller 6701 	<ul style="list-style-type: none"> *Intel 8008, MCS-8, Intellec 8, 8080 *National Semiconductor IMP-8C, IMP-8P Signetics PIP Motorola M6800 RCA COSMAC DEC MPS, PDP-8A Pro-Log MPS-803, MPS-805 Rockwell PPS-8 General Automation (Rockwell circuits) LSI 12/16 	<ul style="list-style-type: none"> Toshiba LCS-12 Intersil CMOS LSI PDP 8-A 	<ul style="list-style-type: none"> *National Semiconductor IMP-16C, IMP-16L, IMP-16P Computer Automation Naked Mini LSI 1 Raytheon RP16 (bipolar LSI) 	<ul style="list-style-type: none"> American Microsystems (used by Frigidaire) Western Digital (used by DEC) Mostek Rockwell (used by General Automation)

¹ Introduced commercially

- Word length.
- Architecture.
- Speed.
- Programming flexibility.
- Completeness. (How many additional circuits are needed to make it work?)
- Available design aids (both hardware and software).

Word length should be the first feature to consider (Table 2). The determining requirements are analog resolution, computational accuracy, character length and width of parallel digital inputs or outputs. Microprocessors are structured for fixed word lengths or for modular expansion by a parallel combination of building-block chips. In the latter category, National Semiconductor and Monolithic Memories have 4-bit "slices" of a CPU, or central processing unit. In some microprocessors the word lengths for addresses exceed those for instructions—for example, General Automation's LSI 12/26, Intersil's LSI PDP 8-A and Rockwell's PPS-4. A large address word eliminates the need to manipulate smaller—4 and 8-bit—data registers to obtain 12-to-16-bit addresses. Higher-speed parallel, rather than time-multiplexed serial, addressing is a resulting benefit. In general, longer word lengths for either addresses or instructions provide higher system throughput and more powerful memory addressing, while shorter word lengths require somewhat less hardware and smaller memories.²

Architectural features include general-purpose registers, stacks, interrupts, interface structure and choice of memories. General-purpose registers are used for addressing, indexing, status and as multiple accumulators. They simplify programming and conserve main memory by eliminating memory buffering of data. Multiple accumulators are especially important for ROM programs that have no writable memory.

Stacks can be used for nesting subroutines and interrupts and for temporary storage of data when programs reside in ROMs. Stacks consist of read/write (RAM) memory locations maintained by software—called a pointer stack—or by registers built into the processor chip—called a hardware stack. The pointer stack—found in the Intel 8080—offers size restricted only by the external RAM, but it must be maintained by software. The hardware stack—found in National Semiconductor's IMP, Rockwell's PPS-4, and Signetics' PIP—is faster, but its size is limited. An additional advantage of the hardware stack is that RAM is not required.

For applications where asynchronous or unpredictable events occur, an interrupt capability is valuable. Throughput increases, since the processor can perform useful work concurrent with I/O (input/output) operations. The major

Microprocessors: Some ABC's

All microprocessors use large-scale integrated-circuit technology. Silicon-gate, p-channel MOS is the most commonly used process. But manufacturers also employ n-channel MOS, silicon-on-sapphire MOS and bipolar processes for increased speed. And they use complementary MOS for lowered power dissipation.

Programmability—that flexible feature not found in random-logic designs—can be obtained on one of two levels. A very detailed level of control is provided at the *micro-instruction* level. These micro-instructions may be used to obtain a *macro*, or machine-language, instruction set, which is then used to write control programs for the microprocessor. New machine-language instructions may be defined by coding new micro-routines. In this way an instruction set can be tailored to an application.

Control programs can also be written in micro-code. This provides increased execution speed and more detailed control at the expense of more difficult programming. Microprocessors that are not microprogrammable contain fixed, general-purpose instruction sets, which are often adequate for most applications.

characteristics of this capability include interrupt latency (time to recognize the interrupt and branch to the service routine), response (time to identify the interrupted device and begin execution of the device service code) and software overhead (to get to the service routine and return to the main program). Single line, multilevel and vectored interrupts offer various speed-hardware tradeoffs. Cascaded interrupt capability (interrupting an interrupt) is essential if slow and fast devices are to be mixed in a system. Interrupt enable flags are used to mask or unmask individual levels.

In a single-line interrupt system, all device-interrupt requests are ORed together to form one request line. The program must identify the device interrupting and resolve priority. This may be done by issuance of a device-select status order, in which all devices report their interrupt status on an assigned bit of the I/O bus. Multiple sense (multilevel) lines are sometimes provided for this function. These lines are interrogated by individual device-select status orders. A vectored interrupt offers very fast response by directly branching to a memory location that corresponds to a specific interrupt. Request/acknowledge and interrupt identification use external hardware, thus alleviating software overhead and increasing speed.

Consider the latency and response times of National Semiconductor's IMP-16C single-line and

Table 2. Applications and characteristics depend on word length

	4-bit Arithmetic or simple control functions	8-bit Controller	16-bit General Purpose	20, 24, 32-bit Special purpose
Typical applications	BCD display control or calculation Electronic cash registers Business and accounting systems Credit card verification Intelligent instruments Appliances Game machines	Intelligent terminals and instruments Data concentrators or front-ends (communications pre-processor) On-board computer (automobile) Process, numeric and machine control Text editing typewriters Traffic control Education— computer science courses or computer design projects Medical electronics Measurement systems	Data acquisition systems—a/d, d/a processing Process monitoring and alarm Supervisory control —gas, power, water distribution Navigational systems Automatic test systems (particularly in-house LSI testing by semiconductor manufacturers) Word processing systems Peripheral control	Digital signal processing—FFT, auto correlation Digital filtering Interfaces to larger computers (wider word length)
Key characteristics	Controller or arithmetic processor Simplified I/O BCD arithmetic instructions Address formation capability Small parts count for minimum systems Low cost and easy to use	Flexible I/O Hardware to reduce software overhead and simplify I/O Multiple addressing modes Interrupt feature Speed		Higher throughput and/or speed requirements Modular LSI building blocks Multiprocessor configurations Special math instructions or routines

vectored interrupts (Table 3). The total time to get to the service routine for the device can be as high as 34.85 μ s for single-line interrupts, but only 4.55 μ s for vectored interrupts.

The choice of memories is important because the memory section often represents a major portion of hardware cost. Read/write memories are commonly used for variable data storage and for program storage during software development. Field-programmable ROMs have become quite popular for program storage in small and intermediate-volume systems and for high-volume prototype systems. MOS pROMs may be erased by ultraviolet light and then reprogrammed. For ease of design, memory modular blocks should be used; the memory-address bus width determines the upper limit of expansion (maximum number of addressable locations). Available microprocessor cards include ROM, pROM and RAM memories and eliminate the need to design or specify a memory system for limited storage requirements.

The interface structure should be easy to use for simple, low-cost applications (parallel or serial). Separate busses for data, addresses, memory and peripheral input and output are most appropriate in this case.

While the type of control depends on the processor, higher throughput results from the use of a direct-memory-access (DMA) bus. In this arrangement, a peripheral device communicates directly with memory without disturbing the CPU. Interfacing is more complex because request and acknowledge signals must be exchanged between the device and an autonomous bus controller. When a single bus is used, data and addresses must be time-multiplexed, and latches must be provided to hold the address stable while memory or a peripheral are accessed.

Provision for handshake I/O control allows convenient interfacing with peripherals of varying response time. Control flags and jump condition inputs are useful to reduce hardware decoding and software overhead. If multiple devices are to be connected over the same I/O lines, three-state or open-collector TTL logic is required to drive the bus. The microprocessor I/O circuitry should directly interface with these signals.

How do you measure speed?

Cycle time, state time, minimum instruction time, time to add two numbers, and interrupt

response time have been given to measure speed. These numbers are rather meaningless, for they do not measure the power of the instruction set. Benchmark programs for a specific task should be coded and the execution times compared to determine which microprocessor meets the speed requirements.

The degree of programming flexibility can be determined from an examination of the instruction set.³ Multiple addressing modes conserve main memory, simplify programming and increase speed through single-word memory-reference instructions. For programs stored in ROM or pROM, indexing or pointer addressing are the only means to access data tables in program loops. Other useful capabilities include bit and byte manipulation, multiply and divide, double-precision arithmetic, normalize and I/O control instructions.

Custom instructions through microprogramming can upgrade performance by optimizing the microprocessor architecture. In some cases other processors, including minicomputers, may be emulated with the microprogram control technique, thereby enabling software built for a larger machine to run on a microprogrammed microprocessor.

The number of additional IC packages required provides an indication of the completeness of the microprocessor set. For example, the Intel 8080, a one-chip microprocessor, requires six logic circuits, nine memory packages and two clocks for a minimum system. To drive more than one TTL load, address and data buffering must be supplied.⁴ In many cases a more powerful multichip processor may be used with no increase in total component count.

Functions generally requiring additional components include clock generation and timing, memory and I/O control, data and address buffering, multiplexer inputs, interrupt control, and sometimes memory refresh control (for dynamic MOS RAMs) and additional power-supply voltages.

What support do you need?

In addition to the microprocessor itself, support should be provided by the manufacturer to simplify the application of the processor and the development and prototyping of the end product. This category includes documented manuals, application literature, area field specialists, prototyping systems—such as National Semiconductor's IMP 8P and 16P, Intel's Inteltec 4 and 8 and Applied Computing Technology's PPS-4MP (for Rockwell PPS-4). Also generally useful are program-development software and the ability to fashion the microprocessor into different configurations.

Table 3. Latency and response times: Keys to speed

	*Latency time— μ s	Response time— μ s	Total interrupt overhead time— μ s
Single-line interrupt	5.95	0-28.9 (1 to 16 devices)	5.95-34.85
Vectored interrupt	4.55	0 (independent of number of devices)	4.55

* Plus time to complete current instruction

Prototyping systems are essential to develop and debug hardware, firmware and software for the end product. The ingredients of such a prototyping system include expanded memory capability, a teletypewriter or card-reader interface, power supply, chassis, control panel and support software (assemblers, compilers, loaders, debug and edit packages). A pROM programmer provides very fast turnaround time when control programs are modified.

With a processor card, you get an assembled and tested microprocessor system, complete with memory. Cards are often more economical than chips for low-volume applications. They may be used for development of early production models for high-volume applications, before they are replaced by an in-house design.

Of course, chips represent the ultimate in low cost and small size, but the designer must interface them with additional components and perform the necessary tests. The components selected must meet exact speed, power and functional specifications.

Manufacturers often point to such features as the number of instructions and registers, memory bandwidth and speed to measure computing power. These criteria are often misleading because they are defined differently by various manufacturers. The only real way to determine the effectiveness of a microprocessor is to code benchmark programs or experiment with representative logic designs.

References

1. Lewis, D.R. and Siena, W.R., "Microprocessors or Random Logic" (three-part series), *Electronic Design*, Sept. 1, 1973, p. 106.
2. Reyling, G., "Microprocessors—Next Generation in Digital Design," *EE Systems Engineering Today*, November, 1973, pp. 86-90.
3. Weiss, D., "Software for MOS/LSI Microprocessors" (three-part series), *Electronic Design*, April 1, 1974, p. 50.
4. Mazor, S., "A New Single Chip CPU," *COMPCON 1974 Proceedings*, pp. 177-180.

Analysis of Microprocessor Instruction Sets

C. DENNIS WEISS, PH.D.
Bell Telephone Laboratories, Holmdel

The use of microprocessors, or MOS/LSI "computers-on-a-chip," requires programming skills. And that may seem to be a disadvantage. Hardware designers once concerned with such matters as latch selection, clock phases and propagation delay must now consider less familiar software-oriented factors like subroutine nesting, indirect addressing and computational algorithms.

However, a review of the basic vocabulary of microcomputer programming can help start you on the way to a microprocessor design. Moreover a review of the differing microprocessor instruction sets can establish a particular microprocessor's capabilities.

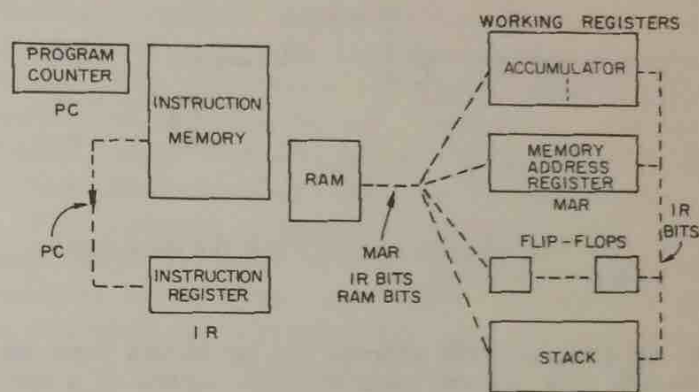
From a programmer's point of view, microprocessor instructions break down conveniently into the following:

- Data movement.
- Data manipulation.
- Decision and control.
- Input/output.

Data can be moved about between a variety of internal sources and destinations. The primary places are shown in Fig. 1. The most complex locations are those in memory—usually a RAM or RAM bank—since a variety of addressing modes can be used to specify location.

The *effective address* of a memory location to be read or written can be given *immediately* by bits in the instruction being executed (Fig. 2). In current microprocessors the immediate data may be 4, 8, 12 or even 16 bits long. Immediate data may be interpreted as a location (or displacement) in a previously selected page (or location) of memory.

The technique of *indexed addressing* permits a 16-bit address to be generated without providing all 16 bits in a current instruction. Were all 16 bits required, the instruction would necessarily be multiworded. The effective address is obtained when the instruction adds immediate data—say, 8 bits—to a designated register usually called the index register.



1. Data flow among the major storage areas is shown by broken lines. Bits in the storage locations control the flow.

In computers the index register may contain fewer bits than the immediate address. Hence the register appears to be a displacement with respect to the immediate address. Though the reverse is usually true with microprocessors, the same view can be taken, since we usually increment the index register to access successive words in memory. The index register can be thought of as a base register plus variable displacement.

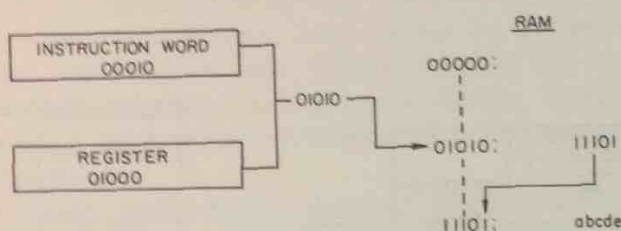
An effective address may also be formed by *indirection*. In this case, a memory address is first computed by use of immediate data or by indexed addressing. Call this a direct address. Then, its contents are taken as the address of the actual memory location to be read or written. This is indirect addressing, a powerful technique that allows any memory location to serve as a memory address register; its content can be used to point to another possibly arbitrary word in memory.

Use addressing modes to advantage

An example follows on the use of various addressing modes (Fig. 3). Assume we write a routine to manipulate data stored in memory locations A_1, A_2, \dots, A_n . All references to these locations are by immediate addressing.

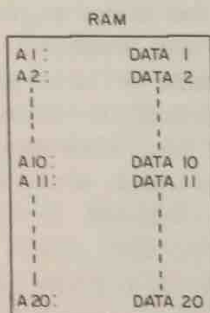
How to compute effective address

1. Use immediate data, given in the current instruction word.
2. Use the contents of the memory-address register, which can be manipulated separately.
3. Add together immediate data and a base, or index, register. This technique is called indexed addressing.
4. Use the contents of a memory location which itself is computed using all of the above. This technique is indirect addressing. An example follows:



The final address is 11101 and the data finally accessed are abcde.

2. An effective RAM address can be formed from immediate data, address register, index register or a combination of techniques.



Computation required:

$$G_1 = F(\text{Data 1}, \dots, \text{Data 10})$$

$$G_2 = F(\text{Data 11}, \dots, \text{Data 20})$$

■ In the program to compute G_1 and G_2 , data must be referred to by addresses.

■ If A_1, \dots, A_{10} appear as immediate addresses in the program to compute G_1 , this program will not compute G_2 .

■ If the program uses a memory-address register to point to Data, the register can be initialized either to A_1 or A_{11} . The program then increments the register to compute G_1 or G_2 .

■ If indexed addressing is available, then a program which computes G_1 will do G_2 if we first add 10 to the appropriate index register.

■ Indirect addressing would provide the most flexibility in this kind of problem.

3. An example of data movement illustrates the effect of different addressing modes.

We can apply the routine to different blocks of data, either by successively loading each block in locations $A_1 \dots A_n$, or by modifying the instructions in the routine to refer to new memory locations. Either alternative can lead to inefficient programming, while the latter alternative is, in fact, impossible if the program is stored in a ROM.

Now consider the case where the original routine used indirect addressing, so that $A_1 \dots A_n$ contain addresses of data. A simple change of the contents of $A_1 \dots A_n$ allows the routine to operate on a new block of data located in a different block of memory. Of course, indexed addressing can be used to achieve the same flexibility.

If the new data locations have the same relative displacements as the original block of data, a reinitialization of the index register allows the routine to access new data. Indirect addressing is not even required in this case. But when the relative displacements are not the same, indirect addressing becomes more useful.

Accumulator—the essential register

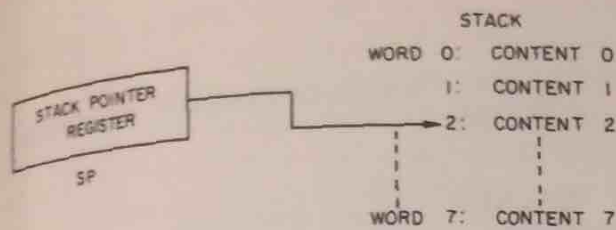
Microprocessors generally have several working registers. However only a single register, usually called an accumulator, is essential, so long as it has access to read/write memory and there are instructions permitting immediate addressing and data manipulation between the accumulator and a memory word. With indirect addressing, even the function of index registers can be accomplished with memory.

The major significance of working registers lies in access time and the bit efficiency of instruction words. It takes far fewer bits to specify one of several previously defined working registers than a memory location. Whether these registers are in an external memory or in the CPU is irrelevant, so long as they can be referenced efficiently. But a faster execution time can be obtained with registers that are separate from memory. They can be accessed for read and write operations without users incurring excessive memory-cycle delays.

The quantity of registers may not be as significant as their quality. For example, can each register be incremented and tested for zero, or is only the accumulator so equipped? If each can, then each register can be used for counting and program loop control.

Which registers can you use for indexed addressing, if any? Can all registers be loaded directly from memory, or can they be loaded only from the accumulator? Which registers can be used as a source or destination for arithmetic/logic operations?

It's difficult to say how many registers are



Current SP value	Operation	Next SP value
2	PUSH	3
7	PUSH	0
2	POP	1
0	POP	7

- TO WRITE data, or PUSH onto Stack, store in location addressed by SP. Then increment SP.
- TO READ data, or POP from Stack, decrement SP. Then read data from location addressed by SP.

4. Last-in, first-out stacks are a common feature of microprocessors. The order of instructions contains address information within the stack.

DATA FORMATS

width: 4, 8, 16 bits, 25 digits
encoding: binary, BCD

ARITHMETIC FUNCTIONS

add { with or without carry bit
between accumulator and register, memory or immediate data
multiple precision possible
possibly with skip if carry out used

subtract - not always.

multiply { by subroutine or special purpose
divide hardware
increment/decrement

LOGIC FUNCTIONS

complement

rotate } with or without extra carry/link bit
shift }

AND
OR
EXCLUSIVE-OR

compare accumulator (with register, memory or immediate data) and skip

5. Data manipulation instructions. Missing instructions may be performed by a subroutine.

needed in general, or even in any particular application, and the number varies widely in current microprocessors. Some have stack-oriented registers that can only be accessed in a last-in, first-out basis (Fig. 4). This orientation is not a serious limitation, since algorithms can often be planned so the required data always are on top of the stack. Stacked registers have the advantage of being more numerous than individually addressed registers. Also, instruction bits

are not required to address them. A stack instruction can refer to only one register, the top register of the stack.

Memory-address registers may be the ordinary working registers, or specially designated ones, such as the program counter. A key register in any computer, this counter points to the next location in memory for an instruction-fetch operation. In addition it's common to have an independently controlled register that points to a read/write memory location. Instructions to load and store the program counter are extremely important, since they permit modification of the instruction sequence. A special advantage results when the counter can be loaded or modified by a value in the accumulator or other working register. This simplifies the control of a program's sequence through computed or external data. Otherwise we would have to rely solely on test-and-branch, subroutine call or fixed jump instructions in program store—where the instructions may not be modifiable.

For example, suppose a microcomputer system must perform certain functions that are selected by an input data word, interpreted as a command for some service. How do we translate this input-bit configuration into the desired computer response? We want to go to a certain program location associated with that command. If we can load the program counter with data, the input command word can be encoded directly as an instruction address. The loading of a portion of the counter causes sequencing to begin immediately at the desired program location.

Alternatively, we can use the input data as an index to enter a table containing program location addresses and load the appropriate address into the counter to cause the desired jump. If program instructions are stored in writable memory, we can modify the address information in a jump instruction before executing it, according to the requirements of the input data.

But if the program is in read-only memory, and the program counter cannot be loaded with data, we must resort to something as complex as the execution of a possibly lengthy decision routine. This routine consists of a sequence of stored instructions that contain all possible desired jumps. Repeated testing of the input data sequences through the decision routine in such a way as to arrive at the desired jump instruction.

One of the most sophisticated addressing modes is found in the National IMP-16. It uses immediate and indexed addressing, either with respect to the program counter or one of two index registers. In addition the IMP-16 permits indirect addressing, either with or without the use of indexing, to compute the effective address. The 256 lower order addresses in the RAM can also

be specified with use of an 8-bit field in the current instruction word.

The simplest data-movement instructions are found in 4-bit microprocessors, such as the Rockwell Microelectronics PPS and Intel 4004. These, as well as the 8-bit Intel 8008 machine, also require separate instructions to load or manipulate a memory-address register, through which all memory references are made. The 8008 contains a single 16-bit memory address register, with 14 bits actually used. The Intel 8080 permits six 8-bit working registers to be used in pairs to provide three 16-bit memory address registers. In addition a 16-bit address for memory reference can be specified by two immediate bytes in certain load and store instructions.

The Fairchild PPS-25 has a unique instruction field for memory references. A mask-programmed repertory of six fields permits assignment of one of six predefined fields in each 25-digit (100-bit serial) register. Only the selected data field is affected by the data movement or arithmetic instruction. A separate program-controlled pointer permits access to any single-digit (4-bit) field.

Data manipulation capabilities

Generally the arithmetic capabilities of microprocessors are limited to addition and subtraction, and usually in a binary format (Fig. 5). The Fairchild PPS-25, however, features decimal arithmetic performed on 4-bit BCD-encoded digit fields. And several other machines include special instructions for handling BCD fields. Apart from the PPS-25, data words vary from 4 to 16 bits, so that multiple-word arithmetic is often required. Care must be taken that carry bits are added into the successively more significant fields—a capability that is always available.

Multiply and divide functions must be performed by subroutines in most systems. Or they can be performed in microcode for microcomputers like the National Semiconductor GPC/P, which are microprogrammable.

Microprocessors, especially those designed primarily for calculator applications, may not allow logic operations. For example, the Intel MCS-4 and Fairchild PPS-25 don't have operations like AND, OR, EXCLUSIVE-OR. However, they do permit complement, shift and rotate operations. The usual rotate or shift is by 1, but the National IMP-16 features rotation by an arbitrary amount in a single 16-bit instruction containing immediate data. The execution time is, of course, a function of the number of shifts called out. However, the instruction bit efficiency is high.

When shift and logic operations are omitted, they can usually be accomplished by a sequence

JUMP
CALL
RETURN } can be conditional or unconditional

BRANCH
SKIP } always conditional

JUMP	Location	Instruction
	k:	i
	k+1:	i+1
	*k+2:	i+2 = JUMP to location m
	m:	j

*At this point, the PC was loaded with m rather than being incremented to k+3.

CALL

Same as JUMP except that PC content is saved so we can return to instruction at k+3. A RETURN instruction performs the restoration.

A user can select either an on-page (short) or arbitrary (long) JUMP address in the Rockwell PPS, Intel 4004 and Fairchild PPS-25.

6. Some instructions change the order in which other instructions are executed.

of other instructions that are available. For example, "shift left by 1" is equivalent to the addition of a binary number to itself. As long as an individual register bit can be tested—say, by rotation into a carry flip-flop—all logic operations can also be performed whether or not individual instructions for them exist. However, considerable additional time will be spent.

Increment and/or decrement—critical arithmetic functions—can be accomplished along with test-and-skip functions. Such multiple-function instructions are particularly useful in controlling passes through program loops. For example, the Rockwell PPS has a 1-byte instruction that adds a 4-bit immediate field—say, the number 1—to the accumulator. If a carryout is generated (when the register reaches its maximum value), the next instruction word is skipped, but the carry flip-flop itself is not disturbed. The National IMP-16 has an analogous 1-word (16-bit) instruction.

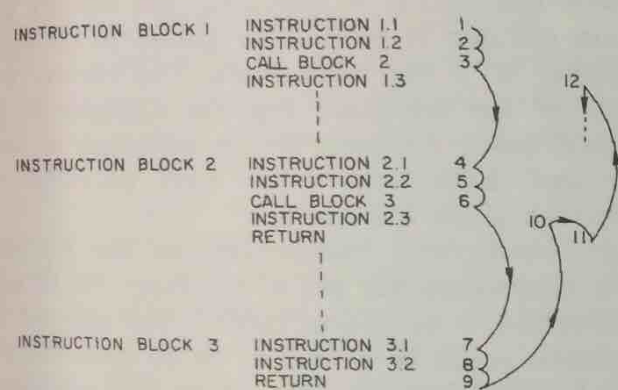
A similarly powerful instruction in the Intel 4004 permits incrementing any one of 16 4-bit registers. If the result is zero, the next instruction in sequence is taken; if nonzero, a jump occurs to an immediate location on the same ROM page designated by the second byte of the current instruction. Again, the accumulator and carry flip-flops are not affected. Here, a 2-byte instruction is used that provides a more flexible jump instead of a skip.

An interesting extension of the increment/

decrement capability occurs in the National IMP-16. A memory location can be incremented or decremented with skip if the contents become zero. This feature permits efficient use of memory locations as counters for control functions. Also, the processor's addressing modes specify the effective address of the memory word to be incremented or decremented.

The Intel 8080 also permits a single memory location to be incremented or decremented. Internal flip-flops are affected, so a conditional jump instruction can be used later to test the memory content for zero.

An unusual and powerful feature of the decimal arithmetic in the Fairchild PPS-25 is the ability—through mask-programmed options—to



7. CALLS are nested to a depth of two in this example. Illustration shows the order of execution.

specify one of six fields over which any arithmetic function is to operate. The words are organized with a maximum of 25 decimal BCD 4-bit fields. Hence part of a register can be treated as a mantissa and part as an exponent. Appropriate arithmetic can be performed on these fields under program selection. Otherwise we would have to mask out or store different data separately. An individual decimal field can also be singled out by reference to a pointer register, which is itself under program control.

Decision and control capabilities

Microprocessors use the common convention of sequencing through instructions in order, unless directed otherwise by a decision-and-control type of instruction. The instruction changes the value of the program counter (Fig. 6).

Microprocessors execute JUMPS, CALLS or BRANCHES. The program counter may be changed unconditionally by a JUMP or CALL instruction, or conditionally, depending on the outcome of a specified test. These instructions are called conditional JUMPS, conditional CALLS or BRANCHES.

The difference between a JUMP or BRANCH

on the one hand and a CALL on the other (conditional or otherwise) has to do with whether or not the program counter is saved. In a CALL, the program counter (or counter plus 1) is saved. Thus the counter can conveniently be restored to point to the instruction that would have followed the CALL had the instruction stream not been changed by the CALL. The RETURN instruction restores the counter to the instruction following the last executed CALL. RETURNS can be conditional, as well. If the condition is not satisfied, the counter is not restored but is simply incremented once again. It then points to the instruction stored after the conditional RETURN.

A further distinction can be made as to the ability to nest CALLS. Such nesting is illustrated in Fig. 7, where a series of CALLS transfers the program counter to a sequence of instruction blocks. By an execution of a series of RETURNS, the counter eventually returns to a location in the original block.

CALLS and JUMPS can be crucial

The use of conditional CALLS and JUMPS is absolutely crucial to programming (Fig. 8). Essentially they allow programs to respond to inputs rather than simply to deliver the same answers to the same programmed questions. A program must do different things, depending on the condition of the machine: Has a carry been generated? What is the current computer result? Is the number zero? (If it is, don't divide by it.) Has an interrupt or new command been issued? And so on.

All microprocessors allow such conditions as "carry bit set?" and "accumulator = 0?" to determine whether or not a JUMP, CALL or BRANCH is to be executed. Some permit branching as a result of logic levels presented on direct input lines, or individual bits in registers, or program-set flip-flops, or register parity, or a stack-full condition, or still other requirements. Again, the absence of one condition can almost always be overcome by the use of extra program steps. In a common situation, JUMP or CALL occurs if a condition is TRUE. But a symmetrical instruction in which the FALSE condition triggers the JUMP or CALL does not exist. By use of an extra unconditional JUMP or CALL, of course, the deficiency can be overcome.

The address loaded into the program counter when an unconditional JUMP or CALL is executed—or when a conditional JUMP or CALL or BRANCH is executed—may be specified in the same variety of ways in which memory is addressed: immediate or indexed direct; or indirect, through a memory location which itself is

TYPES OF CONDITIONS (JUMP, CALL, RETURN)

True or False on

- carry FF
- zero register (usually Accumulator)
- sign (most significant bit) of register
- parity of register
- programmer controlled flip-flop
- test input

If condition fails, do next instruction in sequence.

3-WAY BRANCH CONDITIONS (FAIRCHILD PPS-25)

example: if (A) < (R), PC ← PC + 4 bit immediate data
 (A) > (R), PC ← PC + another 4-bit field
 (A) = (R), PC ← PC + 1

SKIP CONDITIONS

- if (R) ∩ M = 0, skip
 - if (R) > M, skip
 - if (R) ≠ M, skip
 - if Flip-Flop = 1, skip
 - if R (lower) = 4-bit immediate data, skip
- } National IMP-16
- } Rockwell PPS

COUNT AND JUMP/SKIP

- increment R and if ≠ 0, do short JUMP (Intel 4004)
 - increment (decrement) M and skip if zero (National IMP-16)
 - (A) ← (A) + M and skip if carry out
 - (A) ← (A) + 4 bit immediate field and skip if carry out
- } Rockwell PPS

NOTE: (A) = contents of accumulator
 (R) = contents of register R
 M = contents of memory location currently addressed.

8. A summary of conditional instructions shows the variations possible for JUMP, CALL, RETURN, BRANCH and SKIP.

specified by an immediate or indexed mode. The obvious reason for using JUMPS is to get to a new section of the program. For example, all work routines in some systems may report back to a main executive routine by an unconditional JUMP.

Unconditional CALLS allow us to use one copy of a sequence of instructions, a subroutine, and to enter it from many different routines. For example, a multiply subroutine can be called whenever required in any instruction sequence. With a CALL, a single RETURN as the last subroutine instruction causes the program counter to return to the sequence immediately following the CALL. A nesting facility enables the programmer to write subroutines that themselves call on other subroutines to perform operations. Thus several arithmetic subroutines might call a still simpler subroutine that shifts a register a certain number of places.

The Rockwell PPS microprocessor allows unconditional JUMPS to one of 64 locations on the current 64-word page. The locations are specified by 6 bits of data in the 12-bit JUMP instruction. Unconditional long JUMPS provide an immediate 12-bit address in the two successive words of the instruction. The CALL and

RETURN instructions are also unconditional.

The short CALL in Rockwell's PPS is an example of indexed immediate and indexed indirect program-counter addressing. The instruction itself specifies an immediate partial address consisting of the six low-order bits of an address. These bits are indexed by a fixed page address, called page 60. When the directly addressed word on memory page 60 is read, its 8-bit content is used as the low-order bits of the program counter. The high-order 4 bits of the counter are automatically set to 1110. Thus the first JUMP is made to an address given indirectly by any one of 64 locations on page 60.

The final JUMP seeks any one of the 256 locations on pages 56 through 59—pages with 6-bit addresses whose high-order 4 bits are 1110. When the CALL is executed, the current program counter is pushed, or placed in the upper register of a two-level stack. The current contents of the top stack register displaces to the second stack register, whose contents, in turn, are lost. Execution of the next RETURN instruction pops the stack.

All conditional instructions in the Rockwell PPS microcomputer are of the form "SKIP next instruction if condition holds." The skipped instruction could be chosen to be an unconditional CALL or JUMP, thereby giving the equivalent of a conditional CALL or JUMP for the complementary condition.

The Intel 8008 has conditional and unconditional JUMP, CALL and RETURN instructions. The CALL and JUMP use 14-bit, immediate addresses only (and thus a 3-byte instruction), and CALL uses a seven-level stack for pushing and popping the program counter. There is, however, a single byte unconditional CALL instruction that pushes the counter and replaces it with an address consisting of all zeros except for bit positions 3 through 5. These are given as immediate data in the instruction. Hence eight short, but frequently used, subroutines can be located in the lower order 64 locations of memory, accessible by exceptionally fast and short CALLS. The 8008 accepts and executes such an instruction on its input bus upon receipt of an interrupt signal. This feature enables direct control by external devices of JUMPS to routines that handle interrupts.

Microprocessor has stack pointer

The Intel 8080 contains a 16-bit register called a stack pointer, which is incremented and decremented automatically by CALL and RETURN instructions. The current program counter is stored in (for a CALL) or loaded from (for a RETURN) a RAM location whose address is given by the contents of the stack-pointer regis-

ter. This permits arbitrary depth nesting of CALLS. But since the memory locations must be reserved for this use, a limit must be set on the depth.

In the National IMP-16, all registers and condition flip-flops can be pushed or popped from the internal 16-level stack, thus providing a convenient way to save the entire state of the processor. Registers and condition flip-flops can also be saved in or restored from the RAM stack area in the Intel 8080, but not in the Intel 8008. This capability is particularly important in interrupt-handling applications.

An unusual feature of conditional instructions in the Intel 8008 and 8080 is the way in which three of the condition flags—ZERO, PARITY EVEN, and SIGN BIT 1—are interpreted. They refer to the register last referenced by an instruction that might change a condition.

The Fairchild PPS-25 has a very flexible control structure. It uses unconditional JUMPS to an address within the same ROM, as specified by 8 bits of immediate data from the JUMP instruction. These 8 bits are interpreted as a signed-2's-complement number that is added to the address of the current ROM location. The feature permits jumping forward or backward a specified amount from the current location. A separate ROM-select instruction changes the high-order bits of the program counter, permitting a JUMP to a new ROM page.

Conditional JUMPS can lead to either two-way or three-way branches. A two-way BRANCH is an ordinary JUMP instruction. Three-way BRANCHES involve either two different modifications of the program counter (both using immediate data) or execution of the next sequential instruction. A pair of instructions selects the desired conditional mode.

CALLS in the PPS-25 are accomplished in two steps. First, the current program-counter value plus 1 must be stored in one of two fields in a special status register. Then an unconditional JUMP or conditional JUMP or BRANCH is executed. The execution does not itself save the content of the counter. A RETURN is accomplished by reloading the counter with the current content of the appropriate status-register field, again after the counter automatically increments once. This second incrementing ensures a skip over the JUMP or BRANCH instruction that followed the original counter storage instruction.

The National IMP-16 exploits its 16-bit instruction word to permit flexibility in generating the addresses for unconditional JUMP and CALL instructions. The counter is loaded with an effective address that is computed from an 8-bit immediate-data field (a signed-2's-complement displacement) that is added to the 16-bit

content of an index register. If the indirect mode is selected, this address is used to access a memory location whose content becomes the value for the counter. In the CALL instructions the current counter value is saved in a 16-level stack. The RETURN instruction retrieves the counter value from the top of the stack and adds to it 7 bits of immediate data from the instruction itself.

The Intel 8080 has an instruction that transfers the 16-bit content of two working registers into the program counter, thus causing an unconditional JUMP. The JUMP address originally in the working register could have been obtained by a computation or table look-up operation. The National IMP-16 provides this same flexibility, since the effective JUMP address can be based on an index register content. Or it can be based on the content of one of the 256 lower order RAM locations, in which case an indirect memory-reference mode would be selected.

The conditional instruction in the IMP-16 is a JUMP and provides an 8-bit displacement (7-bit magnitude plus sign) that is added to the current counter value. One of 16 condition flags can be tested, including several externally and internally controlled flip-flops.

Input/output capabilities

The nature of the microcomputer interface and the I/O instructions vary considerably from one system to another (Fig. 9).

A basic scheme employed in the Intel 8008, 8080 and the National IMP-16 provides bits on the address bus for both input and output instructions. With an INPUT or OUTPUT enable pulse, these instructions can be used to select an I/O device. Then the microprocessor either puts out the accumulator contents as OUTPUT data or gates the input bus content to the accumulator. The address-bus bits in the Intel machines come from the current instruction word; in the IMP-16, they are more general, being formed by an addition of immediate data from the instruction and the content of an internal working register.

The Rockwell PPS system uses immediate data for device selection, but then it provides a bi-directional data exchange in the same cycle. The 4 bits in the accumulator go out on 4 bits of the instruction-data bus. This is followed by a loading of the accumulator from the remaining 4 bits of the same 8-bit bus. The INPUT instruction for the Intel 8008 also outputs the accumulator before loading it from the main instruction-data bus. Hence every executed INPUT instruction can also be used to output data to the same peripheral address.

The Intel-4004 uses I/O ports that are asso-

Example: Intel 4004

(I/O ports are associated with special ROM and RAM devices bus-connected to the 4004 microprocessor)

Ports

- a RAM output port (4 bit, latched)
- a ROM I/O port (4 bits, mask programmable to specify direction)

Selection

- one (or two) set-up instructions select a ROM and RAM device

Data transfer

- $(A) \leftarrow$ input port bits on ROM
- ROM output PORT bits $\leftarrow (A)$
- RAM output latch $\leftarrow (A)$

Example: National IMP-16**Selection**

- $(R) + 7$ bit immediate field is transmitted as a 16-bit device address/enable, accompanied by an I/O enable signal. It is sent to the Address Register.

Data transfer

- $A \leftarrow$ (external device)
- (external device) $\leftarrow A$

9. Input/output instructions combine a selection and data-transfer operation. These can be triggered by successive instructions or by a single combined instruction.

ciated with the ROM and RAM devices of a complete MCS-4 system. A ROM and RAM are

selected by separate instructions. The I/O port of the ROM—each of 4 bits is mask-programmed as either an input or output terminal—and the latched RAM OUTPUT port can be read or written with an appropriate 8-bit instruction.

The Fairchild PPS-25 uses a set of I/O commands to control special I/O devices designed for use in this system. In addition it contains an unusual direct 8-bit (serial) input to the ROM address register. Data on this input are added to the ROM address register. Also, a special instruction loads serial data into the active status register, where each bit can be interrogated as an individual flag.

Bibliography:

"IMP-16C Application Manual," Publication No. 4200021B, June, 1973, National Semiconductor, Santa Clara, Calif. 95051.

"MCS-4 Microcomputer Set Users Manual," Revision 4, February, 1973, Intel Corp., Santa Clara, Calif. 95051.

"PPS-25, Programmed Processor System Preliminary Users Manual," October 25, 1972, Fairchild Semiconductor, Mountain View, Calif. 94040.

Wickes, W. E., "Parallel Processing System (PPS), Application Notes," Publication 2518-D-17, January, 1973, Rockwell Microelectronics Div., Anaheim, Calif. 92803.

"8008 8-bit Parallel Central Processor Unit, Users Manual," Revision 4, November, 1973, Intel Corp.

"8080 Preliminary Specifications," Revision 1, Intel Corp.

MOS/LSI Microcomputer Coding

C. DENNIS WEISS, PH.D.
Bell Telephone Laboratories, Holmdel

Engineers who incorporate MOS/LSI microcomputers in their designs face a critical need: conversion of system algorithms into instructions that can be loaded directly into the system's memory.

IC manufacturers are giving more and more attention to this phase of design, generally called coding, with improved tools and techniques to simplify the designer's task.

The basic tools available are these:

- Assemblers.
- Editors.
- Loaders.
- Compilers.
- Microprogramming.

Fig. 1 shows the primary function of the first four tools. In addition hardware or software simulators are available for program testing and error locating.

Assembly language: the most appropriate

An assembly language, the most common for microcomputer programming, has these features: symbolic operation codes; labels that refer to memory locations—instruction or data; and symbolic names for operands, such as registers, condition flip-flops and test conditions of conditional instructions (Fig. 2).

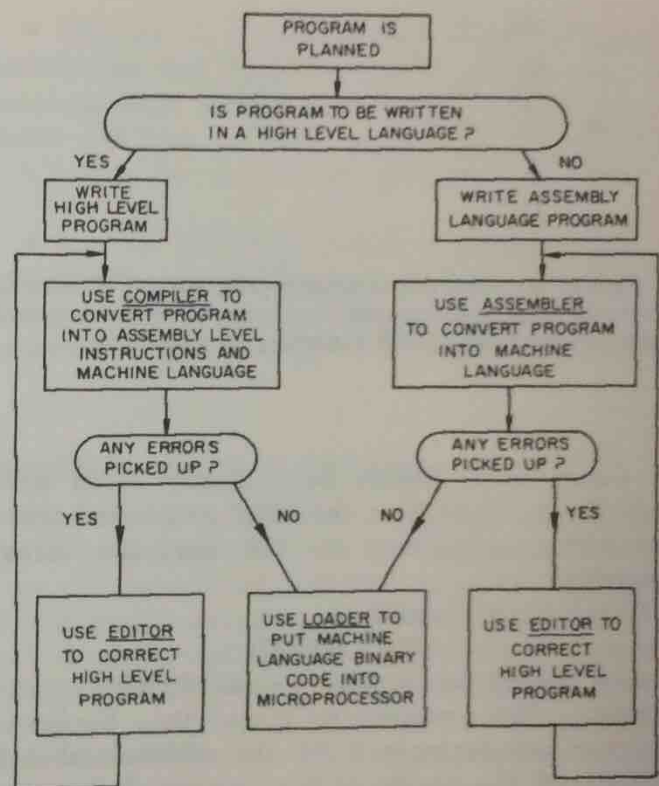
For example, in the Fairchild PPS-25 the instruction¹

$$(R_{1j}) \leftarrow (A_j) + (R_{kj})$$

replaces the contents of register R_1 with the sum of the contents of the accumulator and register R_k . However, only a designated field, j , in each register is involved in the addition. The Fairchild assembly-language equivalent reads

ADD Y, X, T.

Here Y represents the name of a destination register, X the name of a source register and T a previously selected code that represents the field over which addition is to take place. The possible codes of T, with their meanings, include the following:



1. Preparation of the binary code to be placed in read-only memory can be simplified by use of a compiler or assembler and an editor and loader.

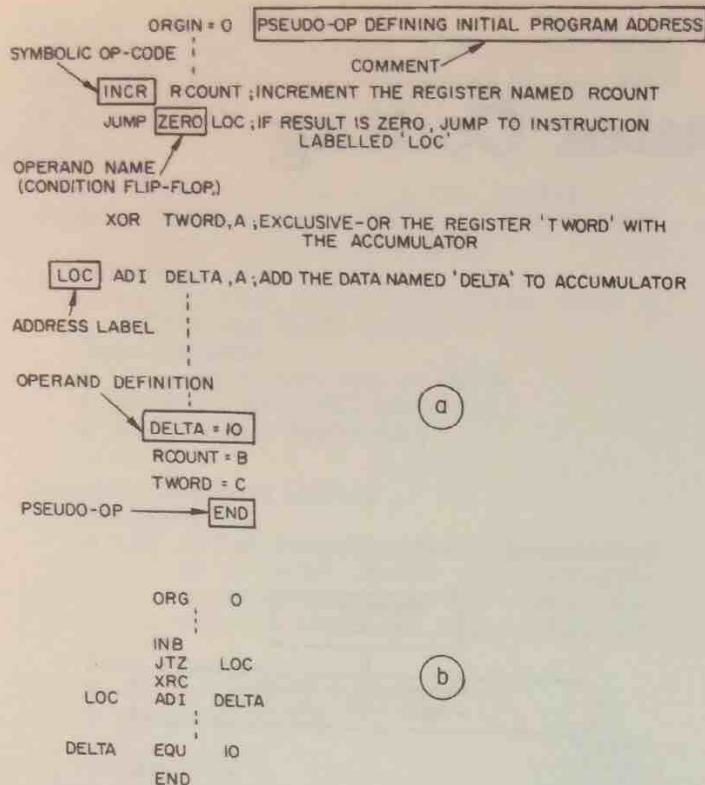
TOTAL: Total field,
FRAC: 19 (left-most) digit fractional or mantissa field,
LSD: Least significant digit,
PFIELD: Digit selected by pointer register.

In the Intel 8008, consider this conditional CALL instruction: $PC \leftarrow S$ and $(PC) \leftarrow 14$ bit immediate field, if condition holds; otherwise do next instruction. PC refers to the program counter and S represents a last-in, first-out stack.

Such an instruction in the Intel assembly language is written

CTX PLACE.

X refers to C, Z, P or S, which mean, respective-



2. Part of an assembly-language program (a) illustrates the basic language features. The same program segment appears in the Intel 8008 assembly language (b).

ly, Carry = 1, Result Zero, Parity Even and Sign Bit 1. PLACE is the label associated with any other instruction in the sequence being assembled.

Hence the statement

CTP STEP1

causes the microcomputer to call STEP1 conditionally. The processor saves the program counter and replaces it by the address labeled STEP1, if the parity of the register last operated upon was even. Otherwise the instruction that follows would be executed.

The sequence

```

INB
CFP STEP1
JMP STEP2

```

increments register B, calls to STEP1 if the parity of register B is odd or performs an unconditional JUMP to STEP2 if the parity is even.

The assembler can read a source tape or file with statements written in the symbolic assembly language (Fig. 3). Also, the assembler can construct various tables from the source file and produce an output object tape, or file, with binary numbers for the microcomputer.

For example, in the Fairchild PPS-25,

ADD B,C, FRAC

appears in the object code as

000100101010.

A line of source tape

```
LD ACφ, @. +10
```

This says LOAD register ACφ indirectly, through the address given by adding 10 (octal) to the current value of the program counter (denoted by.)

A line of the list tape

```
10 000 9109 LD ACφ, @. +10
```

10 = line number in assembly language program (on source tape)

000 = location of the instruction

9109 = hexadecimal representation of the 16-bit machine language word

LD ACφ, @. +10 = assembly language statement written by programmer on source tape

A line of the object tape

```
10010000100001001
```

This is a 16-bit machine language equivalent of the instruction above.

3. The assembler,—a program,—converts a source tape to a list tape and absolute object tape in this example from the National IMP-16.

From left to right, 000 is the operation code for ADD; 100 is the Fairchild code for the B register; 101 is the code for the C register, and 010 represents the mask-programmed code to select the left-most 19-digit field of a register.

For the Intel 8008,

CTP STEP1

appears as the 3-byte instruction,

01111010

00110000

xx001110.

STEP1 is assumed to be an instruction stored at binary location 00111000110000. The last two bytes give, respectively, the low 8 and high 6 bits of the address. The bits marked x are "don't cares" for the 8008. The assembler could substitute any bit pattern, since the machine ignores these locations.

The assembler—a program

The assembler is a program that must be run on some computer. One assembler program—from Intel—can be loaded into several PROM or ROM chips and executed by a microcomputer of the type for which it is assembling. These are called "hardware assemblers," because they run on the hardware itself.

A more common situation is one in which the

```

NUMBERS OCTAL
ORIGIN 0
ENTRY 1 LOAD R1, MEM 1
        LOAD R2, MEM 2
        'LOAD' IS UNDEFINED OP-CODE ***
***
ENTRY 1 COMPARE R1, R2
        DUPLICATE ADDRESS LABEL ***
        JCOND PLACE
        'PLACE' IS UNDEFINED ADDRESS
        LABEL ***
        OPERAND MISSING ***
        JUMP FINISH
        STORE R1, MEM; if R1 > R2, EX-
        CHANGE
        'MEM' UNDEFINED ***
        STORE R2, MEM 1
FINISH  HLT
        'HLT' IS UNDEFINED OPERATION ***
MEM 1   = 1732
MEM 2   = 1840
***
NUMBER IS INVALID OCTAL ***
        END

```

```

NUMBERS OCTAL
ORIGIN 0
ENTRY 1 LOAD R1, MEM 1
        LOAD R2, MEM 2
ENTRY 2 COMPARE R1, R2
        JCOND GREATER, PLACE
        JUMP FINISH
PLACE   STORE R1, MEM 2; if R1 > R2, EX-
        CHANGE
        STORE R2, MEM 1
FINISH  HALT
MEM 1   = 1732
MEM 2   = 2040
        END

```

4. An assembler provides error messages that start with "****" in a program with errors (top). The corrected program appears at the bottom.

assembler itself is written in Fortran. With minor modifications, the program can be run on any computer that compiles Fortran programs. Thus the designer prepares source programs, assembling them on some other computer, to obtain the object tape for the microcomputer. The Fortran-written assemblers are often made available to users through various national time-sharing, computer-service companies.

Assemblers contain pseudo-operations

Assemblers provide more sophisticated features. These are usually pseudo-operations, or assembler instructions, that do not assemble into microcomputer instructions directly but control the assembly of instructions that do. The more significant and common pseudo-ops are as follows:

- **NUMBER SYSTEM (B,O,D).** If B is written, all literals that appear in operand fields are interpreted as binary numbers. Similarly O and D establish octal and decimal modes.

- **ORIGIN.** The statement `ORIGIN 256D` causes the next instruction to be stored at location 256 (decimal). Consecutive locations are used until another `ORIGIN` statement appears.

- **COMMENTS.** It's common to intersperse English text in a source file that contains assembly language. With the selection of a symbol, such as `" / "` or `" ; "` or `" : "`, the assembler ignores all symbols to the right of the selected one on each line of source text. But the assembler reproduces the symbols in the final list file.

- **EQUAL.** A statement such as `R1 = PLACE` establishes that `PLACE`, and `R1` can be used interchangeably as names of register `R1`. The statement `DATA1 = 53D` causes the contents of `DATA1` to be taken as 53 (decimal).

- **DATA GENERATING STATEMENT.** A statement such as `TABLE D 7, 53, 29` creates three data words stored in successive locations in memory. The first location is labeled `TABLE`.

Assemblers give error messages

The ability of assemblers to detect and point to a variety of errors in source statements is one of their most valuable features (Fig. 4). These errors are syntactic—they deal with misuse of the actual language. Assemblers normally cannot catch logic errors in the program, errors of intent or other subtle problems. A statement that contains an error is printed in the list file with a code letter—a flag—beside it. Or the entire error message may be printed.

Some common errors that can be detected include duplicate address label, undefined label and unrecognized instruction mnemonic (due perhaps to the misspelling of an operation code). Other detectable errors include undefined operand field names, wrong number of operands and an invalid number in the number system chosen. In addition an assembler could be made to detect the error of an address referred to the same ROM page, as in a short `JUMP` when a long `JUMP` is required.

Not all errors of syntax are flagged in current microprocessor assemblers. For example, when the labeled address for a `JUMP` or `CALL` instruction is not the start of an executable instruction, the error is not generally detected.

A macro facility—a deluxe feature in assemblers—is very useful when similar sections of

code are used repeatedly but variations preclude the use of conventional subroutine techniques. A macro consists of a sequence of code or a routine that is defined with such parameters as data values, addresses, labels or even instructions. An expansion of a macro involves a specific copy of this sequence in which all parameters have assigned values.

For the assembler to produce an expansion of the macro, only a single statement need be written—if you assume that the macro definition has already been given to the assembler. This statement appears at the location at which the expansion is to begin, and it contains a list of the values to be assigned. The assembler creates the complete expansion where requested.

Editors make changes

Editors are interactive systems that allow designers to prepare a program, or text, and to make changes with simple commands. Time-sharing services, which provide remote access to microcomputer assemblers, have such editor systems. Hence designers can prepare assembly-language programs and correct them. They can add documentation and store, combine and retrieve programs. And they can output programs onto paper tape and printers with relative ease.

Once a program has been written, assembler-flagged errors corrected and a binary object tape, or file, created, the program must be loaded into the memory of the microcomputer system.

Assembled programs can be loaded into mask or field-programmable ROMs. They can also be loaded into RAMs, in which case a small bootstrap loader is required. The latter may be a minimal program loaded into several ROMs or pROMs. This bootstrap program has just enough capability to read an object tape of a complete loader program, which is placed on a tape reader under microprocessor control. More often, the bootstrap loader contains the entire loader program, and all RAM space is available to load the application program.

Application programs can be conveniently tested in RAM before they are committed to ROMs or pROMs. However, if they are to be used in RAMs in the final system, a startup or restart procedure is needed. The procedure permits bootstrapping of the microcomputer into operation. A permanent loader is required in read-only memory.

Advanced loader features

The most elementary binary loader simply reads successive words on the object tape and writes them into successive locations of RAM

A PL/M statement

DECLARE (X,Y,Z) BYTE; IF X > Y THEN Z = X - Y + 2; ELSE Z = Y - X + 2

An equivalent set of assembly language statements for the Intel 8008

	ORG 4000	
BEGIN	LLI LOW X	
	LHI HIGH X	
	LAM;	accumulator contains X
	LLI LOW Y	
	LHI HIGH Y	
	LBM;	B-register contains Y
	SUB;	Subtract B-register from accumulator
	JTS LOC2;	if result negative, jump to LOC2.
LOC 1	ADI 2;	add 2 to accumulator
	LLI LOW Z	
	LHI HIGH Z	
	LMA;	store answer in the location for Z
	JMP FINISH	
LOC 2	LCI 377	
	XRC;	accumulator bits complemented
	ADI 1;	2's complement of X-Y in accumulator
	JMP LOC1	
FINISH	HLT	
LOW X	EQU 70;	word address of X
HIGH X	EQU 10;	page address of X
LOW Y	EQU 71	
HIGH X	EQU 10	
LOW Z	EQU 72	
HIGH Z	EQU 10	
	ORG 4070	
LOC X	DEF 0;	X = 0 initially. Value assigned elsewhere
LOC Y	DEF 0;	Y = 0 initially
LOC Z	DEF 0;	Z = 0 initially

5. A short, readable compiler statement corresponds to many assembly-language statements.

memory. The loader generally starts at a fixed origin. A relocating loader is more complex and not generally available. The reloading loader uses a special object tape and the desired origin data to automatically adjust the program addresses and load the resulting binary instructions.

With a basic binary loader, the same flexibility can be achieved by reassembly of the original source tape or file, but with a change of the origin using a suitable ORIGIN pseudo-operation.

Another feature of more advanced loaders is linking capability. Here program segments or routines with undefined labels or names can be loaded. The loader supplies missing cross references between the separate routines. Again, this feature can be achieved by reassembly of the entire collection of programs.

Compilers translate languages

A compiler is a program that accepts as input data another program, written in a so-called

- Errors in basic system design
 - difference between intended or desired operation and that achieved
- Errors in basic algorithms
 - incorrect algorithm
 - wrong strategy
 - algorithm takes too long to execute
 - arithmetic accuracy or precision unsatisfactory
- Errors in implementation
 - logic error
 - off by one count
 - conditions reversed
 - data stored in wrong order
 - microcomputer hangs up in a loop
 - data destroyed by overstore
 - wrong register used
 - coding errors
 - wrong instruction
- Errors in hardware
 - marginal operation
 - races
 - propagation delays too great
 - wiring error
 - interface signals incorrect
 - peripheral device operated improperly

6. Many potential sources of error exist in a microcomputer design.

source language. The compiler then outputs another program, written in what is called the target language. The latter can be either the assembly language or a machine language.

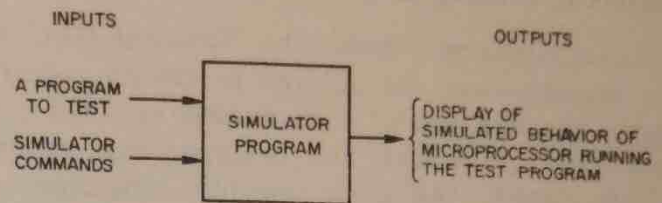
The source language is usually a high-level language, in which the instructions or commands are much more powerful than those of the target language. Examples of source languages are FORTRAN, COBOL, APL, ALGOL or PL/1.

Compilers make the programmer's job easier because they provide a language that requires fewer statements for an algorithm. Compilers eliminate the need to write detailed codes to control loops, to access complex data structures, or to program formulas and functions.

For example, a compiler from Intel has a subset of PL/1 instructions as its source language. The subset language is called PL/M.² An example from PL/M illustrates the powerful nature of the source-language instructions:

```
DECLARE (X,Y,Z) BYTE;
IF X > Y THEN Z = X - Y + 2;
ELSE Z = Y - X + 2.
```

The PL/M statements are converted by the compiler into a sequence of assembly-language instructions. The instructions compute Z after they test to see if $X > Y$. If X is bigger, then $Z = X - Y + 2$ is computed. If $X \leq Y$, then $Z = Y - X + 2$ is computed. X, Y and Z refer to the contents of three, single-byte locations established by the DECLARE statement.



EXAMPLE

INPUT TO SIMULATOR

The program itself (machine language instructions produced by an assembler)

A list of simulator commands

```
SET PC = 0, REGISTERS R1 = 1, R2 = 1, R3 = 1
```

```
SET MEMORY LOCATIONS MEM (100) TO MEM (150) = 0
```

```
STOP SIMULATION AFTER 20 INSTRUCTIONS
```

```
DISPLAY REGISTERS R1, R2, R3
```

```
DISPLAY MEM (100) TO M (103)
```

```
DISPLAY OCTAL
```

```
START
```

OUTPUT OF SIMULATOR

```
R1 = 5 R2 = 10 R3 = 73
```

```
MEM (100) = 0
```

```
MEM (101) = 377
```

```
MEM (102) = 264
```

```
MEM (103) = 113
```

7. Commands to a simulator allow designers to verify that a program is correct.

Fig. 5 shows an equivalent sequence of instructions written directly in the assembly language of the Intel 8008. Notice how much more difficult the instructions are to understand, despite the comments. And notice the increased amount of writing required, even without comments.

The use of higher-level languages has its limitations. Although errors may be reduced because of the lessened detail, new problems can be caused by failure to understand all the conventions built into the compiler. There is also invariably some loss in efficiency in compiler-generated code.

If you rely too heavily on a compiler, your mode of thinking may be too far removed from the actual microcomputer capabilities. While programs are compact, easy to read and much easier to write, the net result may be excessive storage space and slower execution.

One solution is to write routines that are typical for an application in both the compiler's source and assembly languages. The comparison helps to determine any loss of efficiency and how significant the loss may be.

A compiler that produces assembly-language code—and not simply machine-language words—permits the use of an assembly listing for tests and verification. Also, such a compiler lets the designer eliminate redundant data movement.

Microprogramming tailors designs

Some microcomputers—the National GPC/P,³ for example—can be tailored to design requirements through use of a mask-programmed control ROM. In effect, the designer can choose, within limits, the basic machine-language instruction set if he writes the microprogram.

This flexibility simplifies use of a microcomputer as an emulator of another computer. The instruction set of the other computer is microprogrammed into the microcomputer control ROM. Execution of a program instruction corresponds to selection of the equivalent micro-routine.

Microprogramming can also be used for critical, short routines in applications where speed is of the essence. The routines can be executed faster when written in the basic control language of the microcomputer. A single machine-language instruction triggers the routine.

The microprogram instructions are more elemental than the usual machine-language instructions. Each instruction controls limited, simple operations in the microcomputer. A sequence of instructions is required for most machine-language instructions. Hence many instructions are required for an entire computational routine.

Simulator tests programs

Many potential sources of error exist in a microcomputer program of even modest complexity (Fig. 6). A software simulator provides one of the most useful tools for testing programs.

Input data to the simulator consist of an assembled program, or object file, written for the microcomputer. In addition various commands are available to control the simulated execution of the program (Fig. 7).

The simulator output contains representations of the contents of various registers, flags and memory locations. These are shown as they would appear inside the microcomputer. The simulator commands allow designers to obtain selected outputs at simulated instants. A listing of simulator commands similar to those for the Intel 4004 and 8008^{4,5} appears in Fig. 8.

- Start simulation.
- Stop simulation after a given number of cycles of simulated instructions.
- Stop simulation when the processor reaches a specified instruction or memory location.
- Stop simulation when the contents of a specified memory location are altered.
- Display any registers, flags, program counter, stack contents, I/O ports, or memory locations specified in a command and range-list.
- Trace the simulated microprocessor by displaying elements such as registers whenever an instruction is fetched from the memory region specified in a range-list.
- Display the number of instruction states used by the microprocessor since the last simulator initialization.
- Set specified memory locations, registers and I/O ports to specific values to initialize a run.
- Interrupt the simulated microprocessor and force a CALL instruction.

8. A variety of simulator commands is available to test microcomputer programs.

- Hardware exercisers
- Test programs for RAMs
- Logic subroutines for microcomputers which do not have basic logic type instructions
- Decimal arithmetic routines
- Transcendental function routines
- Data format conversion routines
- Teletype or tape drive interface programs

9. Program libraries contain frequently used programs.

As with all computer systems, microcomputer program libraries are beginning to form, with contributions from vendors and users. A brief listing of frequently used programs appears in Fig. 9.

References:

1. "PPS-25, Programmed Processor System Preliminary Users Manual," October 25, 1972, Fairchild Semiconductor, Mountain View, Calif. 94040.
2. "A Guide to PL/M Programming," July, 1973, Intel Corp., Santa Clara, Calif. 95051.
3. "General Purpose Controller/Processor (GPC/P)," Publication No. 4200005A, National Semiconductor, Santa Clara, Calif. 95051.
4. "MCS-4 Microcomputer Set, Users Manual," Revision 4, February, 1973, Intel Corp.
5. "MCS-8, 8008 Simulator Software Package," November, 1972, Intel Corp.

Traffic-Light Controller

An Example

C. DENNIS WEISS, PH.D.
Bell Telephone Laboratories, Holmdel

How does the basic software for a MOS/LSI microcomputer differ from that for larger computers? Not fundamentally. The differences are in emphasis. With microcomputers, you must give more attention to such things as program size, execution times of critical routines and real-time interactions with peripheral devices.

A traffic-light controller provides a good example of how a microcomputer is programmed for a specific application.

Usually the basic algorithm must be selected first. Then working routines can be identified and common memory specified. A generalized program for coding can be easily adapted to any specific microcomputer.

In the traffic-light controller in Fig. 1, the goal is to feed a maximum number of cars through the intersection in any time period. The maximum duration of a red light for each route represents the primary constraint.

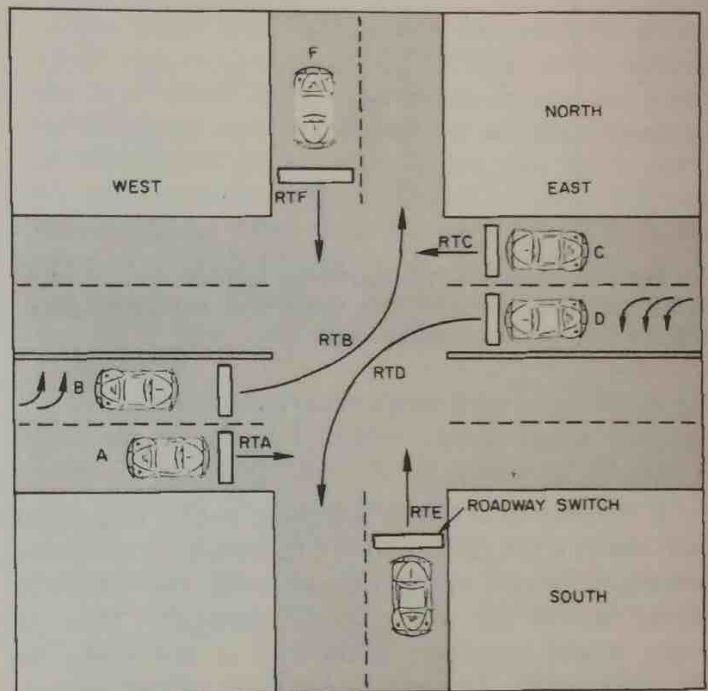
Fig. 2 shows the traffic lights that are visible to cars on each route. The lights are controlled by loading latches, which drive relays inside the traffic-light units. The through-traffic and left-turn parts of the display in Fig. 2a can be set independently to red, yellow or green.

Vehicles that cross switch plates on the roadway actuate switches that can be sampled by the microcomputer. The input port contains six input lines, each of which carries a ONE or ZERO logic signal. Each time an axle passes over a switch plate, the switch toggles and the logic level changes. The logic level is maintained until the next axle toggles the switch.

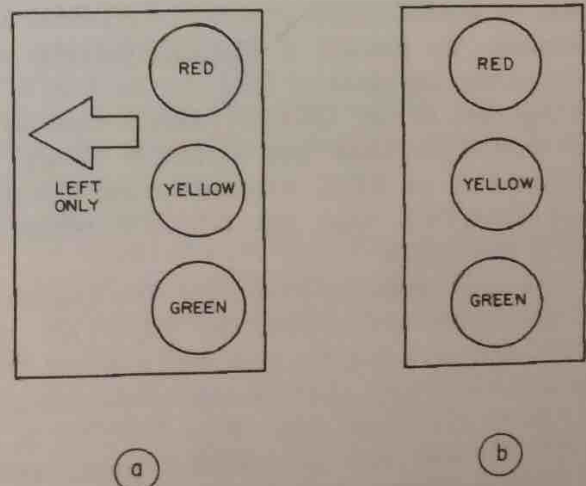
Assumptions clarify problem

The following six points clarify the design problem:

1. The duration of GREEN signals for each ROUTE must be varied to optimize traffic flow, because traffic varies during the 24-hour day.



1. Traffic-light intersection contains six routes: four through lanes and two turn lanes. The six roadway switches toggle each time an axle crosses a switch plate in the corresponding route.



2. Traffic signals under microcomputer control consist of east-west lights (a) and north-south lights (b). The left-turn signal in east-west lights can be independently set to red, yellow or green.

Since the roadway switches cannot provide information on the length of a queue, the controller algorithm must contain a delay; the number of vehicles crossing a switch during the current GREEN cycle determines the duration of the next GREEN cycle.

2. Two toggles of a switch indicate one car has passed. Vehicles with more than two axles represent, therefore, more than a single car for

STATES	ROUTES					
	A	B	C	D	E	F
1	0	1	0	1	0	0
2	1	1	0	0	0	0
3	1	0	1	0	0	0
4	0	0	1	1	0	0
5	0	0	0	0	1	1

3. The five valid STATES for traffic signals. A numeral 1 entry signifies a ROUTE has GREEN in the corresponding STATE; a zero implies RED.

traffic measurement.

3. Traffic on a ROUTE should not be measured for the full GREEN cycle. Otherwise an unstable situation would exist. Lengthening the GREEN cycle would let more traffic through. This in turn would require lengthening it still more on the next cycle. Instead use a fixed initial period, BASETIME, during the GREEN cycle of each ROUTE to measure the traffic. Each GREEN cycle must, of course, last a minimum of one BASETIME.

4. The duration of a GREEN for any ROUTE consists of BASETIME plus DELTATIME, both in seconds. To satisfy a MAXREDTIME constraint on the duration of RED for each ROUTE, limit the sum of the DELTATIMES used by all those ROUTES that get GREEN while one ROUTE holds at RED. Also constrain the number of ROUTES that get GREEN while one ROUTE has RED.

5. The only states allowed for the traffic signals are those in which the ROUTES with GREEN cannot interfere with one another. Obviously east-west and north-south lanes cannot be GREEN at the same time. Also ROUTE B and ROUTE C will not be GREEN concurrently. This feature avoids wasted time while cars yield, start and stop.

6. There is enough real time to do the job with a microcomputer. The only possible time-

critical task involves a count of toggles of the roadway switches. Assume that vehicles travel at less than 60 mph and that minimum axle spacing is 48 inches. These figures yield a minimum of 45 ms for the time interval between toggles on each ROUTE. Hence a microcomputer is certainly fast enough.

A stable STATE for the intersection is defined by specifying which ROUTES have

CONTROL STATE SETS	ROUTES					
	A	B	C	D	E	F
{1,3,5}	1	1	1	1	1	1
{2,4,5}	1	1	1	1	1	1
{2,3,4,5}	2	1	2	1	1	1
{1,2,3,5}	2	2	1	1	1	1
{1,3,4,5}	1	1	2	2	1	1
{1,2,4,5}	1	2	1	2	1	1
{1,3,5}, {5}	1	1	1	1	2	2

4. Row integers give number of GREEN intervals for a ROUTE. State sets labeling each row constitute the control sequence for the GREEN intervals.

GREEN; the remainder are RED. Transitions from GREEN to RED occur between stable states.

From point 5, the only stable STATES are those five tabulated in Fig. 3. Note that exactly two ROUTES have GREEN lights in each stable STATE.

Define a valid sequence

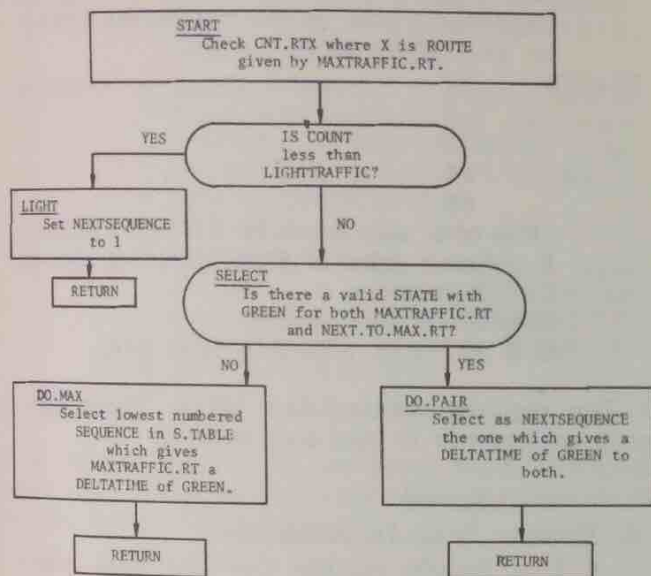
A valid sequence of STATES is one in which each ROUTE gets some GREEN time from a cycling of the sequence. Fig. 4, based on Fig. 3, lists sets of states from which valid sequences can be made. Sets {1,3,5} and {2,4,5} are equivalent, minimum "covers" of the table in Fig. 3, or solutions to the classical set-covering problem.¹

For simplicity, and to satisfy the constraints of point 4, let's establish in advance a repertory of valid SEQUENCES to be used. The ones chosen (Fig. 5) are all those that give a single BASETIME interval of GREEN to each ROUTE and a DELTATIME to exactly two ROUTES or to none.

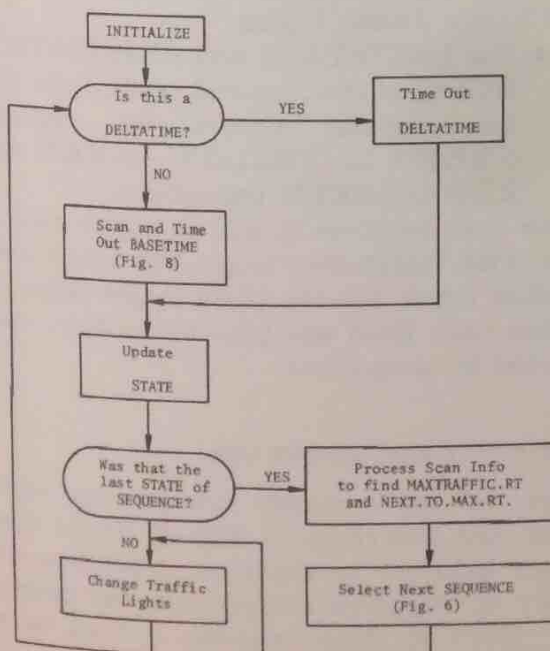
In Fig. 5 the nonequivalent sequences represent sequences that are ordered to eliminate unnecessary light changes. For example, if SEQUENCE3 were changed to STATE1, STATE3, STATE2, STATE5, then ROUTE B would see

SEQ1:	STATE2.	STATE4.	STATE5
SEQ2:	STATE2.	STATE3.	STATE4. STATE5
SEQ3:	STATE1.	STATE2.	STATE3. STATE5
SEQ4:	STATE1.	STATE3.	STATE4. STATE5
SEQ5:	STATE1.	STATE2.	STATE4. STATE5
SEQ6:	STATE1.	STATE3.	STATE5. STATE5

5. The ordering of preferred sequences eliminates unnecessary light changes. STATES shown bold provide DELTATIMES to two ROUTES.



6. Flow chart selects a SEQUENCE of traffic-light STATES. Traffic counts for the two ROUTES with most BASETIME traffic determine the selection.



7. The complete controller algorithm contains the Select Next SEQUENCE algorithm of Fig. 6.

GREEN, RED, GREEN, RED, resulting in wasted start and stop times. The sequence, STATE2, STATE4, STATE5 is chosen over a sequence based on {1,3,5} so that both east-bound or west-bound lanes will move together.

The algorithm that selects a new SEQUENCE from Fig. 5, based on traffic during BASETIMES of the previous SEQUENCE, appears in Fig. 6. The algorithm allocates DELTATIMES of GREEN to the two busiest ROUTES, if possible. If this is not possible, it favors the busiest ROUTE in conjunction with a through ROUTE. Also, if traffic is light—the busiest ROUTE has less than a preset threshold—no DELTATIME is allocated, since this would not result in greater vehicle throughput but only in longer RED times.

The value for BASETIME is normally given, while the choice of DELTATIME must satisfy the MAXREDTIME constraint. From the given BASETIME and an analysis of all possible pairs of SEQUENCES in Fig. 5, you can easily solve for the maximum allowed DELTATIME. Express the resulting RED time for each ROUTE in the form $N_1 \text{ DELTATIME} + N_2 \text{ BASETIME}$. N_1 is the number of DELTATIMES that occur during the RED time, and N_2 is the number of BASETIMES.

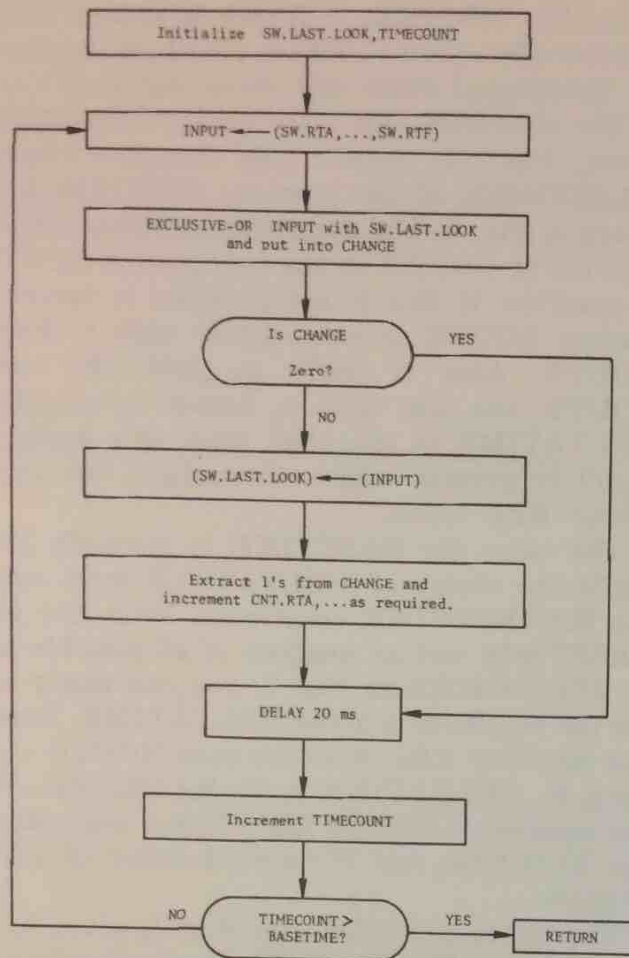
The entire controller algorithm is given by the flow chart in Fig. 7, which incorporates the basic Select Next SEQUENCE algorithm of Fig. 6. If the state of the intersection provides DELTATIME, then you simply create that delay with a program loop. Otherwise scan the toggle switches to count toggles and thus measure the traffic during the BASETIME.

When the current STATE is completed, the STATE must be changed to the next one in the current SEQUENCE. This changes the traffic lights. If the SEQUENCE is completed, a new one must be selected before a return to the start of the major loop.

The working routines

It is now possible to identify the routines to make the controller algorithm work. There are six principal ones:

1. Initialize:
 - Select a starting SEQUENCE, SEQ1.
 - Set all traffic counts to zero.
 - Set other pointers and counters to zero.
2. Time Out:
 - Do a loop of NOPS (do nothing but count) for a DELTATIME period.
3. Scan and Time Out (see Fig. 8):
 - Scan toggle switches every 20 ms, and count the 20-ms intervals until the BASETIME has been spent.
 - Update toggle counts for each ROUTE



8. Scan and Time Out routine, one of the working routines, is outlined in this flow chart.

S.TABLE.POINTER	X
/An address in the S.TABLE whose content is the current STATE in the current SEQUENCE/	
CNT.RTA	X
CNT.RTB	X
CNT.RTC	X
CNT.RTD	X
CNT.RTE	X
CNT.RTF	X
/Counts of number of toggles (of switch in that ROUTE) since last initialization/	
SW.LAST.LOOK	X
/Last toggle positions read for all toggle switches/	
TIMECOUNT	X
/Location incremented once every 20 ms until BASETIME exceeded. More than 8 bits required/	
LASTSTATE	X
THISSTATE	X
MAXTRAFFIC.RT	X
NEXT.TO.MAX.RT	X
/X = 1,2,4,8,16,32 for RTA, ..., RTF/	
NEXTSEQUENCE	X
/X = 1,2,3,4 or 5/	

9. Some common memory locations and content description. Data format for each memory location X must be specified throughout.

S.TABLE1	2	/2 means use STATE2/
	4	
	5	
END1	0	
S.TABLE2	2	
	203	/Use STATE3 as a DELTATIME state/
	4	
	5	
END2	0	

/The SEQUENCE TABLE contains octal numbers defining the sequences of STATES in each SEQUENCE. A "1" in the left-most bit flags that STATE as the DELTATIME STATE for that SEQUENCE/

STATES	12
	3
	5
	14
	60

/Five octal data words for STATES 1 through 5 defining pairs of ROUTES which can be GREEN together.

FORMAT of 8-bit word:

0 0 RTF RTE RTD RTC RTB RTA/

10. Two required constant-data tables. These can be stored conveniently in read-only memory.

as required.

4. Process Scan Information:

- Use toggle counts for the just completed SEQUENCE to determine the busiest ROUTES, MAX.TRAFFIC.RT and NEXT.TO.MAX.RT, as required in Fig. 6.
- Initialize route counts CNT.RTA, ... and CNT.RTF to zero.

5. Select Next SEQUENCE (Fig. 6).

6. Change Traffic Lights:

- Use LASTSTATE and THISSTATE to determine the required traffic-light control signals. Create the required GREEN-to-YELLOW-to-RED and RED-to-GREEN transitions.

Note that routines 2 and 3 create real time delays with instruction loops. The much shorter execution times outside these major delay loops—which vary from one pass to another—can be neglected in comparison.

Plan memory and register use

Some of the common storage locations are labeled and described in Fig. 9. These may be memory locations or registers, depending on the number of registers available in the microcomputer. Each of these locations is used by more than one working routine.

Two fixed-data tables, shown with labels and memory contents in Fig. 10, could appear in

- START
- Get MAXTRAFFIC.RT and shift word until a 1 is found.
Bit1=1→RTA, ..., Bit6=1→RFT.
Let RTX be found.
 - Compare (CNT.RTX) against the constant (LIGHTTRAFFIC).
If (CNT.RTX) is less, go to LIGHT.
Otherwise, go to SELECT.
- LIGHT
- (NEXTSEQUENCE)←1
 - RETURN
- SELECT
- Compute (MAXTRAFFIC.RT)U(NEXT.TO.MAX.RT) and store in MAXPAIR.
 - Compare MAXPAIR against every data word in STATES table (Fig. 10).
If match found, go to DO.PAIR.
If no match, go to DO.MAX
- /NOTE: If a match occurs, let I be the line in STATES table at which it occurs. Then STATEI will give the DELTA-TIME to the two ROUTES./
- DO.PAIR
- Search S.TABLE2, ... until a byte is found with bit8=1 and low order 7 bits match line I of STATES table. Let this byte be found in S.TABLEJ.
 - (NEXTSEQUENCE)←J
 - RETURN
- DO.MAX
- Search S.TABLE2, ... until a byte is found with bit8=1 and a 1 in the same position as in MAXTRAFFIC.RT. Let this byte be found in S.TABLEK.
 - (NEXTSEQUENCE)←K
 - RETURN

11. A "word" program for the Select Next SEQUENCE routine (Fig. 6) can be converted easily to the assembly language of any microcomputer.

ROM. S.TABLE defines the SEQUENCES chosen in Fig. 5. STATES represents the information in Fig. 3 used by the Select Next SEQUENCE routine. If the S.TABLE.POINTER is an 8-bit storage location, the S.TABLE of Fig. 10 should be positioned to fall on a single 256-word page. In this way only the 8-bit pointer has to be incremented to sequence through the STATES. This approach is usually efficient with 8-bit microcomputers.

Instead of a detailed instruction sequence for a particular microcomputer, Fig. 11 shows a general "word" program for the Select Next SEQUENCE routine. The flow chart for the routine is in Fig. 6, which also contains the program labels. The word program can be converted easily to the assembly language of any microcomputer.

Reference:

1. McCluskey, E. J., "Introduction to the Theory of Switching Circuits," McGraw-Hill, New York, 1965. (see Prime Implicant Tables)

Microcomputer I/O Capabilities

ANDRE G. VACROUX

Bell Telephone Laboratories, Holmdel

Most buyers of microcomputers are dazzled by the intricacies of CPU-chip design, but the usefulness of a microcomputer depends closely on its ability to exchange data with peripheral devices. A word to the wise: Explore the I/O architecture before you buy.

A microcomputer's I/O architecture breaks down into these areas:

- Transfer techniques.
- Instruction formats.
- Busses.
- Bus structures.
- Interrupt schemes.
- Memory-access techniques.

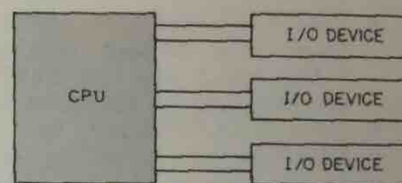
Three kinds of I/O transfer techniques

Most microprocessors allow for three types of I/O transfer techniques—programmed transfer, interrupt-program control and hardware control. In the first two cases, found in most simple applications, the microprocessor controls the transfer. In the third case, system hardware controls transfer.

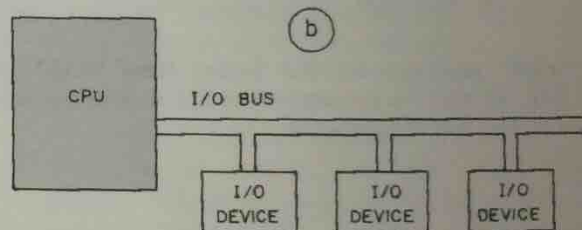
When all I/O operations are under program control—with all instructions to receive or transmit information included in the program—data are transferred whenever the corresponding instruction is executed.

To transfer data, the program addresses a peripheral device with an input or output command. In some cases the program must first check the availability of the peripheral by checking its status and waiting until it is ready. Typical of this approach are applications where information is entered one character at a time—as from a keyboard. In such cases the microprocessor must spend significant “overhead time” waiting for the data to be entered. This isn't a disadvantage in desk-calculator applications, in which the CPU does not have other functions to perform. But it might not be acceptable in a real-time monitoring system.

The interrupt-program approach requires a



(a)



(b)

1. **I/O bus structures employ several schemes.** A radial system is the simplest, but it limits the number of I/O units (a). A party-line system reduces the number of lines needed for a distributed system (b). The latter system also comes in a daisy-chain version, which connects devices serially.

smaller I/O overhead than that of programmed transfer. I/O devices can signal the microprocessor by an Interrupt whenever they are ready to transmit or receive information. When information is received and identified, the microprocessor interrupts its normal program, stores its state and jumps to a subroutine that allows it to perform the transfer operation. Once the interrupt has been serviced, the microprocessor returns to the state at which it was interrupted or some other predetermined state, and it resumes its normal operation.

This approach allows the microprocessor to spend a minimum of time servicing an I/O device. Hence it can perform more operations or handle more peripherals.

Hardware control of information transfer was not used much in early microprocessor applications, but most newer CPUs can accommodate it.

The method requires a significant amount of additional hardware, since the I/O device must initiate and control the data transfer directly into or from microcomputer memory.

But the software support is minimal. It is limited to the initiation, termination and recovery aspects of the transfer. These aspects are performed automatically without microprocessor intervention.

The hardware-control approach, also known as direct-memory access or data break, can be used to transfer blocks of characters directly between a peripheral device—such as tape, cassette or floppy disc—and the main microprocessor memory.

I/O instruction formats differ

The handling of programmed I/O operations varies significantly from one microprocessor to another. Most microprocessors have special I/O instructions of varying length. But some don't have any; the I/O ports are treated as if they were RAM locations.

One of the simplest examples of a special I/O instruction is that of the single-byte instruction, with a different word for each I/O port. Typical is the I/O instruction format of the Intel 8008:

01 RRM MM1.

The five RRMMM bits define 1 of 32 (2^5) possible I/O operations, where RR = 00 implies one of eight input operations and RR \neq 00 one of 24 output operations.

The Mostek 5065 has two types of single-byte instructions. One provides the usual I/O operations for 16 input and 16 output ports:

Input accumulator command 01 10 XXXX

Output accumulator command 01 00 XXXX

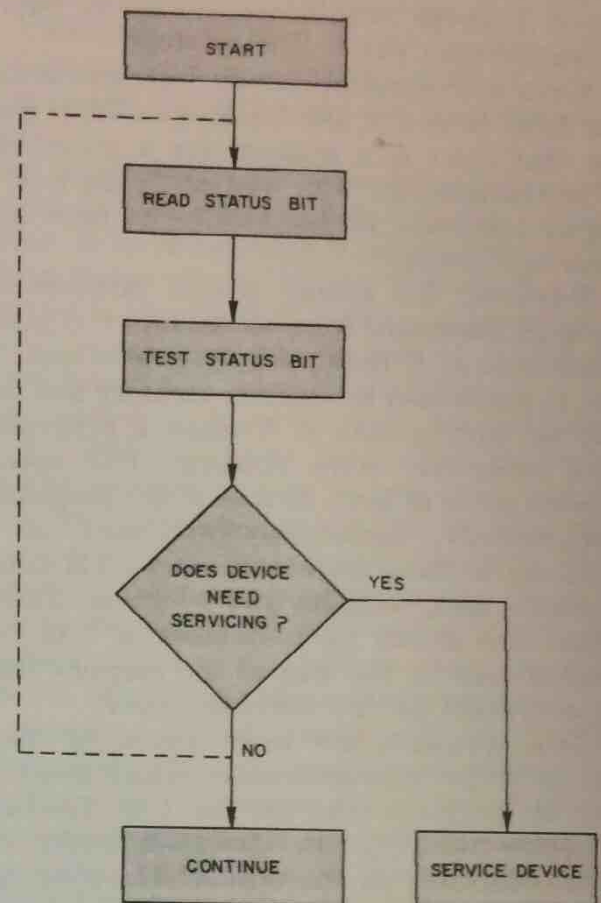
The second type has this form:

Input accumulator skip 01 11 XXXX

Output accumulator skip 01 01 XXXX

During the execution of these I/O instructions (which can be used to access either the same or different I/O ports, depending on system configuration), the CPU tests a flag bit, which may be controlled by the addressed peripheral. Whenever the flag bit is a ONE, the next two bytes of instruction are skipped. This option simplifies the dialogue between CPU and peripheral. Depending on a peripheral's state of readiness, the program can perform an immediate branch.

Despite the extreme simplicity of the I/O instructions for the Intel 8008 and Mostek 5065, this approach limits the number of I/O ports that can be addressed. With the 8008, the number is 32; with the 5065, it's 64. In addition, 1/8 (32/256) or 1/4 (64/256) of the possible instruction words are used for I/O alone. Hence few combinations are left for other purposes.



2. **Peripheral devices can be polled periodically** to find if any need service. However, this simple technique can be time-consuming.

Some microprocessors use a multibyte I/O instruction, although here, again, there are significant variations. Intel's 8080, for example, employs a 2-byte I/O instruction with the following form:

1 1 0 1 X 0 1 1
A A A A A A A A

The first byte specifies an input or output instruction (depending on the value of X). The second byte distinguishes between as many as 256 input or output devices. Hence a few combinations of instructions allow the use of many I/O ports. However, twice as many bytes of control memory are needed.

A different 2-byte I/O instruction is found in the Rockwell PPS-8. The microprocessor is designed to operate with up to 16 performance-enhancing I/O devices, each of which has two 8-bit ports. Software controls the devices, and internal registers store control and status information. The I/O instruction has this form.

0 1 0 0 1 1 1 0
A A A A X C C C

where the first word indicates an I/O operation, AAAA defines one of 16 I/O devices, X specifies

an input or output operation and CCC determines which register within the device is being accessed by the CPU.

From a comparison of the I/O instructions of the Intel 8080 and the Rockwell PPS-8, you can see that there is a tradeoff for a given number of instruction bits. The tradeoff is the total number of I/O ports vs the intelligence built into the interface devices.

However, it's almost always possible to use memory addresses for I/O devices. I/O ports are considered as if they were RAM locations; an input is performed by reading memory and an output by writing into it. Though a program may look somewhat more obscure (I/O operations become more difficult to spot if the program isn't documented), operations performed on input data can be those associated with RAM data. For example, add, compare and test bits. This technique also allows for a number of I/O devices, limited only by the size of the memory that can be addressed by the microprocessor.

This approach has been chosen by Motorola for its M6800 microprocessor, which doesn't have any instructions reserved for I/O. The number of bytes for I/O operations—typically one to three—depends on the type of operation and on the addressing mode. Special peripheral circuits in the M6800 family—such as the Peripheral Interface Adapter or the Asynchronous Communications Interface Adapter—are designed to be compatible with this approach.

The new National PACE processor doesn't have any special I/O instructions either. Like the Motorola M6800, it relies entirely on the addressing of I/O ports as if they were memory locations. Hence all memory-reference instructions can be used to perform I/O operations.

Information travels on busses

Parallel lines and control logic, referred to collectively as the I/O bus, transfer information between microprocessor and I/O devices. The bus contains three types of lines: data, device address and command.

Data lines consist either of one bidirectional set or two unidirectional sets. In the latter case, one set is used exclusively for inputting of data to the CPU and the other for outputting of data. In most cases the width of the bus—number of lines—equals the word length of the microprocessor.

Device-address lines are used to identify I/O devices. The theoretical maximum number of available address lines changes significantly from one microprocessor to another. It depends on the way I/O operations are handled. The number of I/O ports can vary from 32 (or 2^5 , as in the Rockwell PPS-8 or Intel MCS-8) to 65 k (or 2^{16}

as in the Motorola M6800 or National IMP-16).

Command lines allow a peripheral to indicate to the CPU that it has finished its previous operation and is ready for another transfer.

Other lines are also present. You can find interrupt lines on which devices request service, enable or disable lines that can be used to control the interrupt, as well as lines that provide timing whenever required.

The different busses are frequently combined on the same lines to simplify construction and, in some cases, to reduce costs. However, this may increase the number of control lines. The extra lines are needed to extract the necessary information from the common bus.

Three ways to structure the I/O bus

I/O bus structures can take three different forms: radial, party-line or daisy chain (Fig. 1).

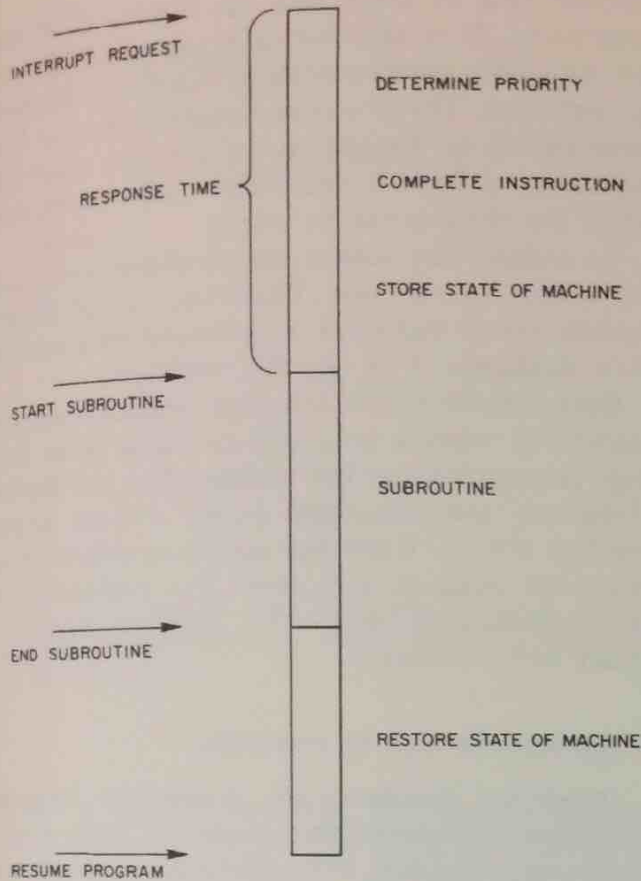
A radial-bus system connects each I/O device to the microprocessor through a dedicated set of lines. It does not allow the connection of more than one I/O unit. Because of its simplicity, a radial bus provides a convenient solution, although it isn't usually compatible with the limited number of CPU pins. However, it is a possibility with the Rockwell PPS-4 system.

A party-line bus is time-shared for data transfers between the CPU and many I/O devices. It must provide means of identifying which device is being called on at a given instant. It does not allow the simultaneous use of more than one I/O unit. All devices are accessed in parallel, and the choice of one or another is controlled entirely by the microprocessor. This bus structure would be justified mainly in the case of a distributed system, since it would significantly cut the number of required lines.

A daisy-chain bus is very similar to the party-line, except that the connections are made in serial fashion. Each unit can modify the signal before passing it on to the next device. This approach is used mainly for signals related to interrupts or polling circuits. Whenever a device requires service, it blocks the signal. A priority is thus established, since the devices that are closest to the microprocessor have the first chance to request service.

The Fairchild F-8, for example, uses the daisy-chain concept to organize its interrupt priorities. Each RAM or ROM chip—which also provides I/O ports—can accept one interrupt input. And each chip can connect to its neighbors to establish priorities. The daisy-chain technique is also used in the Rockwell PPS-8.

Generally a system's bus structure depends on the CPU used. Pin-limited, first-generation CPUs have a single bus that must be time-shared between memory addresses, instructions, input and



3. An ideal interrupt-service routine automatically saves the state of the microcomputer and then restores it after the interrupt has been handled.

output data, device addresses and control signals. This time-sharing requires involved peripheral circuitry, consisting of numerous latches, multiplexers and timing circuits. Also, output information has to be latched before it can be directed toward the appropriate output device—usually another latch. Hence output bus structures usually have to be of the party-line type.

In second-generation microprocessors more than twice as many pins are available. Typically there is a bus for addresses and another for instructions and data, and most control signals are directly accessible. Although some time-sharing still is needed, there's no need for two-stage buffering between the CPU and output device. Nevertheless I/O busses employ a party-line configuration.

Moreover more microprocessors are allocating one or more pins for external flags. For example, the National IMP-16 has two flag bits, while the newer PACE chip offers four external flags. The Mostek 5065 has one external flag. All of these flags simplify programming when a single bit of information has to be exchanged.

Interrupts need servicing

Some applications require that a peripheral device be serviced as soon as possible after some external condition has occurred. In some cases,

especially when the microcomputer is not very busy, this can be done by program control. But most frequently it's necessary to establish some sort of interrupt structure that allows asynchronous external events to change the processing sequence.

When interrupt facilities are not available, the only way to find out whether a device requires servicing is to interrogate it periodically by inputting a status bit and testing it. When the need for service is identified, the program branches to a special subroutine, at the end of which the program returns to its regular operation.

This technique is quite easy to implement (Fig. 2). But significant time could elapse between the moment service is requested and the moment the processor recognizes it. The time can be lessened if the program sits on a small interrogating loop (dashed line in Fig. 2) or, if the microcomputer is programmed to interrogate the inputs frequently. Neither case, however, represents efficient use of a microcomputer.

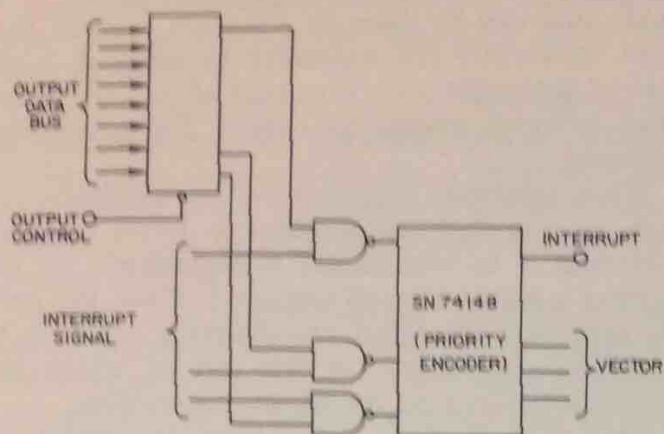
To eliminate wasteful loops without sacrifice in speed, most microprocessors have at least one interrupt input. Whenever an interrupt occurs, the microprocessor terminates the instruction it is executing and branches immediately to a service subroutine (Fig. 3). Ideally the subroutine should do the following:

- Save the microprocessor "state"—all the information contained in the accumulator, the registers and the internal flag flip-flops. (This operation isn't always simple.)
- Acknowledge the interrupt signal on a special line, when it is available.
- Perform the operation called for by the interrupt.
- Restore the state of the machine.
- Resume execution of the program.

The elapsed time between interrupt and the start of the interrupt-handling subroutine is called the "response time." The difference between the total time elapsed and the actual execution time is referred to as the "overhead." Both times should be kept as low as possible.

Interrupt capabilities vary

The capabilities of microprocessors can vary considerably in the way they save the state upon receipt of an interrupt request and restore this state upon completion of servicing. For the Intel 8008, for instance, an extensive amount of software is required. And additional hardware is necessary for saving, at least temporarily, the accumulator and one of the registers. You could avoid the external circuitry by reserving two of the seven internal registers exclusively for status



4. Either an enable/disable function or a priority structure can be obtained readily for a microprocessor that has neither. A conventional input port can be used to control the interrupt input.

saves. But speed and program efficiency probably would be impaired.

Newer microprocessors, such as the Intel 8080, Motorola M6800 and National PACE, have special instructions that save the state of the microcomputer by pushing status information into a push-down, or last-in first-out, stack.

For those applications that have few interrupt sources, the Mostek 5065 offers a unique solution: It incorporates three independent sets of accumulators, program pointers and link flip-flops. Whenever an interrupt occurs, the processor can simply shift from one level of operation to the next, thus making status saves and restorations unnecessary.

Recent microprocessors—such as the Intel 8080, Mostek 5065, Motorola M6800 and Rockwell PPS-8—have Interrupt Enable and Interrupt Disable instructions that set or reset an internal interrupt-control flip-flop. These allow the disabling of the interrupt request, whenever necessary. In microprocessors not having this feature, the only way to achieve the same result is to use external hardware to gate the interrupt signals. The hardware, in turn, can be controlled by a conventional output (Fig. 4).

The Mostek processor employs two special instructions to control the enabling or disabling of its interrupt. The first has the form

0 0 0 0 1 0 $M_1 M_0$,

which allows a designer to enable either Interrupt 1 (M_0) or Interrupt 2 (M_1) or both, by making the appropriate bit a ONE.

The second instruction has the form

0 0 0 0 1 1 $M_1 M_0$,

which allows a designer to disable either Interrupt 1 (M_0) or Interrupt 2 (M_1) or both, by making the appropriate bit a ONE.

The PACE microprocessor has a status register that reserves 6 of its 16 bits for interrupt

control. One of these bits can disable all of the interrupts. It is automatically set to a ZERO by the interrupt service routine, but it can be reset by software. The five other interrupt-control bits each enable or disable one of the four interrupt inputs or a built-in interrupt that is generated when the stack is full or empty.

To control the status bits, however, you must use a few instructions. These load one of the accumulators or registers with the information and then duplicate it in the flag registers.

Each source of an interrupt signal is usually associated with a program-controlled Arm flip-flop. A programmer can enable (Arm) or disable (Disarm) one interrupt source without affecting the others. Until the recent introduction of improved support circuitry, this feature could be implemented only with external hardware under output control.

Assigning priorities to interrupts

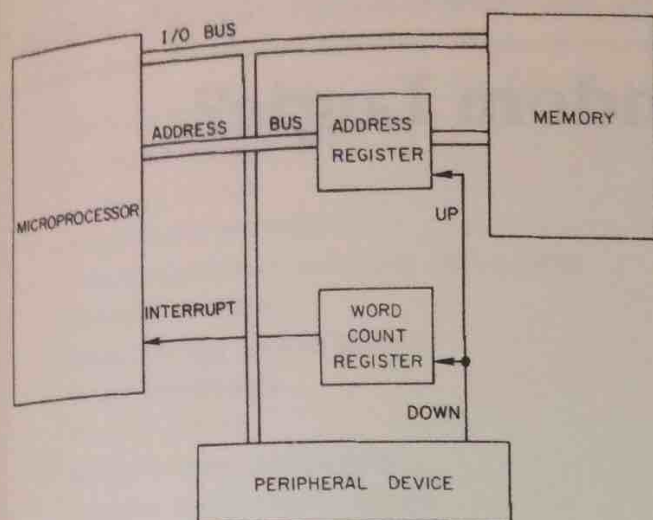
Interrupt requests are frequently assigned priorities. Whenever two interrupts occur simultaneously, the one with the higher priority is considered first. Furthermore a higher-priority interrupt can interrupt the service routine of a lower-priority interrupt. Most microprocessors don't have built-in priorities, and these must be handled either with software, external hardware, or both.

Among the exceptions are the Mostek 5065, which obtains two levels of interrupts through two pins. Also, National's PACE assigns priorities to its four interrupt inputs, and so does the Toshiba TLCS-12 to its eight interrupt inputs. Of course, the daisy-chain structure in Fairchild's F8 or Rockwell's PPS-8 automatically provides priorities.

Most microcomputers have a single-level interrupt: The interrupt causes a transfer of control to a preassigned memory location that contains the beginning of the programmer's interrupt-processing routine. When more than one device may cause the interrupt, the program must poll all possible sources to determine which requires servicing.

For some microprocessors—for example, Intel's 8008 and 8080—the interrupt is "vectored." Whenever these units receive an interrupt request, the microprocessor immediately interrogates a few input bits (the vector). These bits specify one of several addresses—typically eight—and the program jumps to these to find the appropriate service subroutine. Vector interrupt makes polling unnecessary whenever the number of interrupt sources is smaller than the number defined by the vector.

In some cases—the Intel 8008 and 8080—the vector must be constructed with external hard-



5. A direct-memory access facility permits efficient transfer of large blocks of data between memory and peripheral device.

ware that encodes the eight interrupt conditions. This can be achieved easily with priority encoders like the SN74148. In other cases—such as the National PACE—the vector is automatically constructed within the CPU.

Multiple-level interrupts are not found very frequently in presently available microprocessors. The exceptions include the Rockwell PPS-8, with three levels (one of which is dedicated); the Toshiba TLCS-12, with eight levels; the Mostek 5065, with two levels, and the National PACE, with four levels. Multiple-level interrupts allow the microprocessor to determine immediately which device is requesting an interrupt. At the same time multiple levels simplify assignment of interrupt priorities by eliminating the need for special hardware or software.

Many applications require the fastest possible transfer of large amounts of data between the microcomputer memory and peripheral devices. System efficiency can be increased by avoidance of time-consuming programmed word transfers in which the microprocessor supervises each operation.

Increased efficiency can be achieved by addi-

tion of a direct-memory access (DMA) facility. It allows an I/O device interface to "steal" a memory cycle from the program and transfer a word of data directly from or to a memory address specified in a special address register. With an automatic increment of the address register after each word transfer, successive words of data can be transferred into successive memory locations.

A separate, word-count register keeps track of the progress of the transfer. Typically the register is loaded at the beginning of the operation with the number of data words to be transferred and decremented after each transfer. On reaching zero, the word-count register signals the completion of the transfer operation by generating an interrupt signal.

Circuitry initiates memory cycle

Additional control circuitry is also required to initiate the memory cycle, once the data are ready to be transferred (Fig. 5). This circuitry depends on the CPU used. Although most 8-bit CPUs have DMA capabilities, the problems of implementation can vary significantly from one unit to another.

Direct-memory address can be initiated either by a peripheral device or by the microprocessor. In either case programmed control loads the address register with the address of the first memory location, and the word count register with the total number of words to be transferred.

With the Intel 8080, a Hold input can be used to request the CPU to enter a state in which the following occurs: The data bus and the address bus go to their high impedance state, thus allowing an external device to gain control of that bus. The CPU acknowledges the Hold input with an acknowledge signal on its HLDA pin.

In the Mostek 5065, the same result is obtained, respectively, with WAIT (input) and DMA (output).

The most efficient way to implement DMA is given by the Rockwell PPS-8. A special, additional chip can be used to control up to seven independent DMA operations.

Microprocessor or Random Logic?

DONALD R. LEWIS and W. RALPH SIENA
Automata Systems Corp., Kew Gardens, N.Y.

They're called MOS/LSI "computers on a chip." And they're giving designers a new systems building block to replace or upgrade random-logic systems. They're microprocessors, and they're growing in number commercially.

Among other advantages, microprocessors permit a tradeoff of software for hardware to achieve increased system capability and versatility. They can perform many functions and efficiently handle multiple inputs.

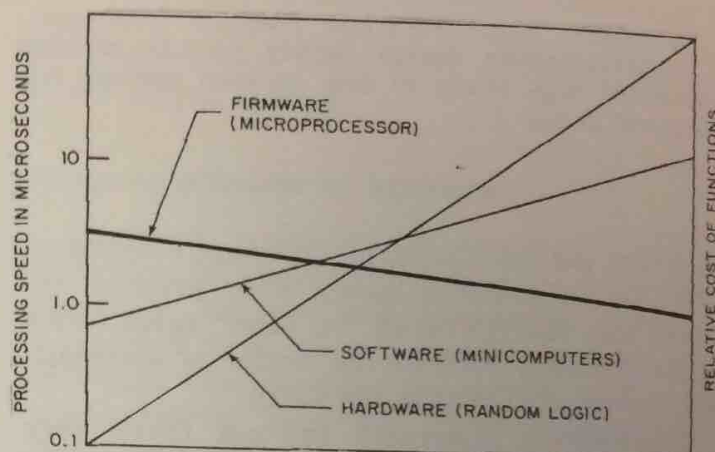
But there are disadvantages, too.

When compared with a random-logic design, microprocessors are much slower. Their initial use requires designers to grapple with relatively unfamiliar disciplines—primarily software. And the complexity and wide-ranging capabilities of microprocessors demand increased system design to ensure that the over-all design functions properly. The result: Choosing between a microprocessor or random-logic approach for complex logic systems requires a careful analysis of the tradeoffs.

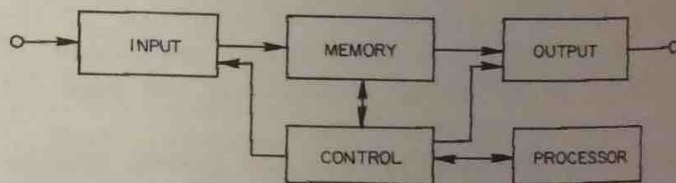
Before the introduction of microprocessors, complex logic systems used discrete and random logic to perform the necessary functions. Integrated circuit families, such as TTL and ECL, developed small and medium-scale integrated functions that simplified random-logic designs. General and special-purpose computer manufacturers used such devices to build their systems. Now third-generation computers use large numbers of these devices, coupled with various types of separate memory systems, to complete their architecture.

Microprocessors find growing uses

However, this computer architecture is too cumbersome and costly for most large digital systems, compared to microprocessors. Such systems include CRT terminals, point-of-sale and other table-top equipment, as well as lightweight airborne equipment. A more complete listing of microprocessor applications is shown in Table 1.



1. Microprocessors offer the lowest cost, but at the lowest speed, with system control provided by firmware. For the highest speed, but at the highest cost, random logic is the way to go. Minicomputers fall between the two approaches in this speed-cost tradeoff.



2. The basic architecture of a microprocessor. This functional diagram covers a wide range of industrial and process-control systems.

A new direction was launched with the emergence of calculator chips. The present calculator can be defined as a small, highly specialized computer. The memory structure consists of both a fixed and a variable memory. The fixed portion, a read-only memory (ROM), provides a system control program called firmware—meaning non-changeable instructions. This contrasts on the one hand with general-purpose computers programmed by software, and on the other hand with random-logic systems that use hard-wired circuitry. The tradeoffs for the three are indicated in Fig. 1.

An extension of calculator design has been the

Table 1. Typical applications for microprocessors.

Desk-top computers
 Automatic typesetting
 Inventory control
 Point-of-sale terminals
 Telecommunication switching and control
 Chemical analyzers
 Manufacturing control systems
 Smart instruments
 Machine control
 Multiprocessor minicomputers
 Adaptive traffic-control signals

Processing oscillographic data
 Banking terminals
 Automobile diagnostic testers
 Intelligent data terminals
 I/O channels for large computers
 Medical electronic systems
 Low-cost radio navigation equipment
 Optical character recognition (OCR) devices
 Automated test fixtures
 Automatic time clocks and payroll systems

development of larger word-length systems that are closer to true computer architecture. This evolution has resulted in the microprocessor, for computation and control applications besides calculators.

Most microprocessors are 8-bit machines, while calculators use 4-bit word lengths but are flexible enough to handle longer words. In addition parallel and serial machines are available, so that a variety of memory configurations can be used.

Basic considerations in system designs

Regardless of which approach is taken—random logic or microprocessor—the design of a system calls for a preliminary evaluation of the requirements. Some of the general considerations are as follows:

- Functions to be performed.
- Amount of hardware required.
- Timing specifications.
- Memory requirements.

The number and type of functions to be performed determine the basic architecture of the system. Systems that operate continually on new data can be built easily and cheaply with random logic, especially where the decisions are few and simple. But in systems with related functions, which require arithmetic, logic control or decision-making operations, microprocessors are the way to go. For systems requiring a knowledge of past operations to perform succeeding operations, microprocessors allow a greater reduction in hardware.

Generally any system that can be laid out functionally like a computer (Fig. 2) can use a microprocessor as the basic building block. The basic computer architecture allows continuous and repetitive use of a minimum of hardware to perform a maximum of functional operations. And the use of semiconductor memories boosts efficiencies, thanks to simplified memory addressing.

Also, systems that can operate on a bus structure for data flow further permit microprocessors to minimize hardware.

Hardware requirements determine the physical size of the system. An estimate of the amount of hardware needed can be determined by answering questions like these:

- How many input and output channels are required for data acquisition and transmission?
- Do all input and output channels use the same data rates?
- Are all input and output channels handling equal amounts of traffic?
- Do input and output channels operate serially or in parallel?
- Are the input and output channels randomly selectable or do they operate in some predetermined sequence?

Based on a detailed analysis of these questions, a preliminary layout of the system should be made, with both random logic and microprocessor circuitry. The differences in the hardware required will become apparent, and for some systems, the differences will be startling enough to point to substantial savings with a microprocessor approach.

System expansion needs should also be kept in mind during this phase of design. Often first estimates of hardware requirements are conservative, because design details are not available. With a random-logic approach, addition of hardware for increased capabilities may not be possible without a complete redesign. But with microprocessors, the expansion can often be readily accomplished by minor changes in the software.

Timing requirements can pose special problems when microprocessors are used. They are slower than most computers and nowhere near as fast as random-logic circuits. To determine these requirements, consider: How much time can be allotted to service input and output devices be-

fore data is lost?

For example, microprocessors cannot supply the continuous output data for a CRT display—especially one that is continuously changing. However they can supply updated information to an output device that services the CRT. But high-speed data channels have to be serviced so often that they can consume an excessive percentage of the over-all processing time.

The over-all timing includes the percentage of time required for each operation (allowing for maximum and minimum values). In some cases the system requires buffer storage of input or output data to meet timing specifications.

Other questions to be answered for a timing analysis include the following:

- How soon does data have to be available?
- How quickly do system functions have to be performed?
- What are the system priorities—which channels have to be processed immediately or more often than others?

Both random-logic and microprocessor systems require some memory storage. A checklist for this part of the design should contain the answers to the following questions:

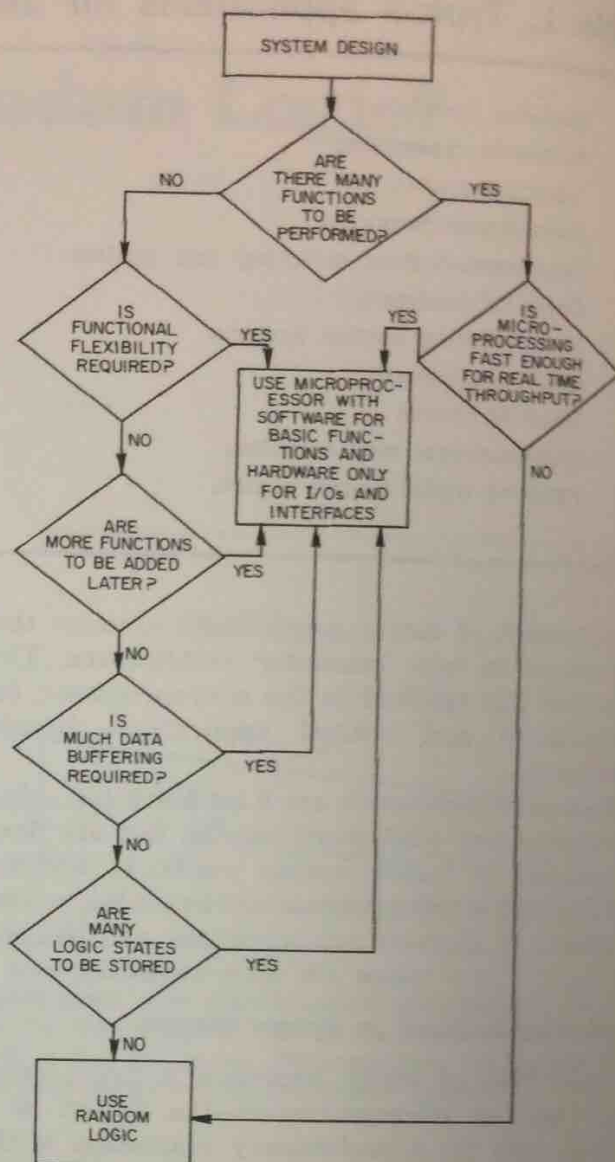
- How much variable information is required?
- What type of programs are to be used?
- Can programs be selected from such storage as tape or disc or be handled as firmware?
- How many file registers are required?
- How many flag registers are required?
- Are other memory uses unique to the system required?

Large memory systems make the use of random logic very unwieldy. Standard microprocessor chips may not be the answer either. In those cases you may have to design a high-speed processor, using small and medium-scale-integration logic ICs, such as those found in TTL families.

Microprocessors vs random-logic

Once the system-design specifications have been determined, the selection of either random logic or microprocessors can be made. While there is a range of applications where either will do, at the extremes one approach is clearly superior to the other (Fig. 3). Random logic offers design advantages when one or more of the following are true:

- The functions to be performed are minimal.
- The input and output consist of single channels.
- The system operates on only one function at any time (though there may be multiple inputs), or the system has a single-word transmission structure.
- A small system has to be custom designed.
- High-speed operation is required.



3. Choose between microprocessors or random logic. A flow diagram can be used to analyze the tradeoffs.

Microprocessors offer advantages over random logic when one or more of the following are true:

- Software can be traded off for additional hardware, so that system capabilities can be expanded readily without system redesign.
- Multiple inputs are needed.
- A large number of functions must be performed.
- Multidecision paths are required.
- Large memories are involved.

The disadvantages of random logic are, not surprisingly, related to the advantages of microprocessors. A random-logic system requires substantial hardware increases for multiple inputs and outputs, or to line up data, or when multiple decisions are required for a given output. Moreover many operations require separate logic for each operation, and variable data must be stored before the required function can be performed—as in arithmetic operations.

Many of the disadvantages of microprocessors are confined to their initial use. Increased development costs and a new learning cycle for designers, for example, are nonrecurring. Generally the use of microprocessors—which really are multiple subsystems—requires system considerations at all design levels. And their use involves a wider variety of design disciplines. Of course, once a design is completed, these aspects with their problems are understood, and thus, no longer disadvantages.

Types of microprocessors available

Available microprocessors can be classified in two ways: by the size of the data-word length by which they perform their processing, and by the type of processing used—either serial or parallel. The most common word lengths are 4 and 8 bits. Some manufacturers state that the word length can be expanded, in multiples of 4 or 8 bits, by combining processor chips.

Four-bit chips are especially useful for systems that perform many arithmetic operations. Originally designed for simple calculators, these circuits later evolved to perform more complex mathematical functions—such as trigonometric or exponential functions. The 4-bit processors can also be used for larger word lengths. However, the words must be composed and formatted with external, and generally cumbersome, hardware.

Most microprocessors, including those expected shortly, are 8-bit circuits. These are designed for terminal or stand-alone operation, although they are not limited to this use.

Most data transmission, primarily asynchronous, requires data-word lengths of 8 bits or less (Table 2). The longer word lengths permit the use of standard codes, such as ASCII, EBCDIC and BAUDOT. And operation with standard codes simplifies the interface with other equipment, like teletypewriters or the more common types of computers and modems. Moreover standard or special codes with up to 8-bit word lengths allow the use of full alphanumeric keyboards. Display outputs are more easily handled, too, particularly where decoders of more than 4 bits are required.

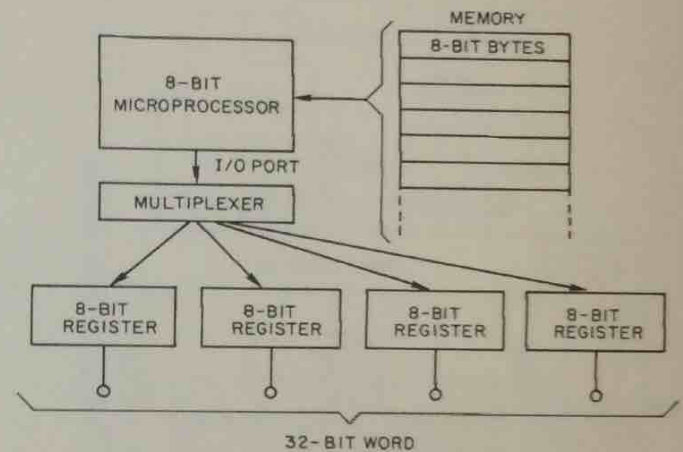
Usually computers of more than 8 bits are preferred because of their greater addressing range and flexibility. Microprocessors are not presently available in word lengths of more than 8 bits. However, manufacturers say that many features of the longer word-length machines can be achieved through the use of external registers and microprogrammable logic.

8-bit processor yields 32-bit word

A typical configuration for implementing a

Table 2. Data processing involves different codes.

Code	Character Length (bits)	Applications
BCD	4	Calculators
BAUDOT	5	International teletype transmission
BCDIC	6	Second generation computers
USASCII	7 (parity optional)	Data transmission standard
EBCDIC	8	Third generation computers



4. An 8-bit microprocessor produces a 32-bit word with external logic. Each 32-bit word uses 4-bytes of memory and four time cycles.

longer word length is shown in Fig. 4. A partitioned word is extracted from memory in 8-bit segments. These words are supplied via the microprocessors to the multiplexer, which routes them to the external registers where they are recomposed and stored. The register outputs now present a longer word length to circuitry external to the processor.

This technique is necessary to achieve greater word length, at the expense of cycle time, where the system does not use an external microprogrammable CROM (control read-only memory). The more general approach taken by some manufacturers is to combine several of their chips in parallel to achieve a greater word length. However, to date, only one manufacturer has presented a system with this capability, and that is accomplished through the use of a CROM.

The second characteristic of microprocessor categorizing is method of processing. Serial processing generally uses a shift-register memory and has the advantage of less hardware. Al-

though the memory shift rate may be operating with a higher speed clock, the access time becomes longer.

Random-access ability is not usable in serial memories. The fetch and execute times are longer. Adding this to the longer access time seriously restricts their application. Where multiple inputs and outputs are used they become impractical. The ability to jump from one part of memory to another is also extremely limited.

Parallel processing overcomes the limitations of serial processing. Parallel processors use a bus for the transfer of data. The bus allows multiparallel paths for data transfer through the system. The fetch and execute cycles, operating on parallel-bussed data, can operate faster. And the use of random accessing of memory is more easily accomplished with a data bus. The waiting time is minimized and the ability to jump from one location to another much simpler to implement with little or no loss of time.

The Anatomy of a Microprocessor Chip

DONALD R. LEWIS and W. RALPH SIENA
Automata Systems Corp., Kew Gardens, N.Y.

The design of microcomputers—computing systems using microprocessor chips—begins with the processor chip. Internal operation, timing relationships and external circuitry are all important because, among other things, they affect the efficient use of the instruction set provided by the manufacturer.

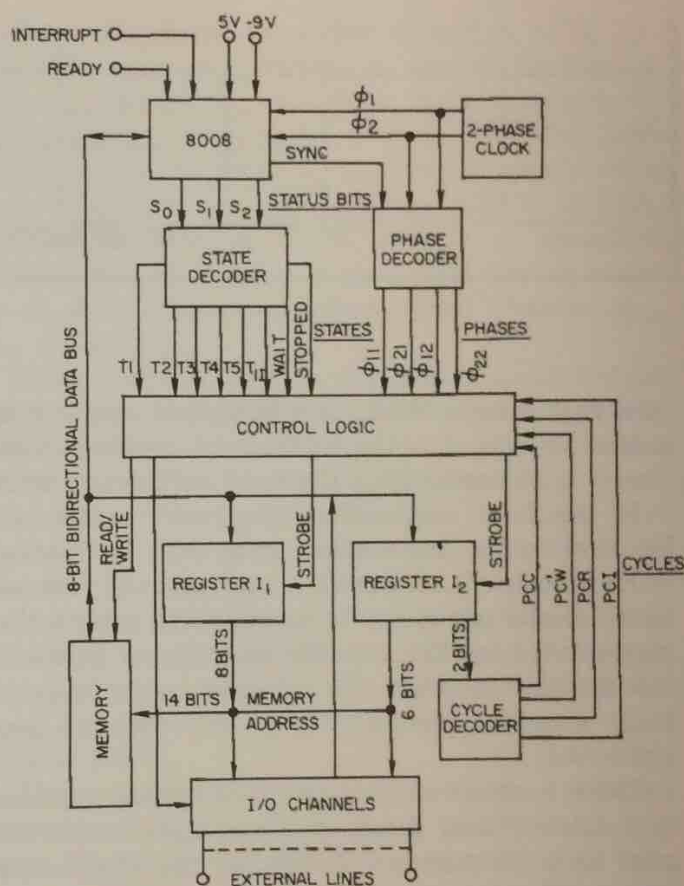
Unlike less-complicated ICs, microprocessors cannot be completely characterized on a simple data sheet. An internal microprogram controls the complex circuitry of microprocessors. And its operation depends on software that you must provide. But you won't know how unless you carefully analyze how the chip works.

Let's look at an 8-bit parallel microprocessor—the Intel 8008, which has been second-sourced. A single-chip MOS/LSI processor, the 8008 can directly access up to 16,384 bytes of external memory using a 14-bit address. The seven 8-bit general-purpose registers on the chip can accommodate an ASCII character or two BCD digits.

Two supply voltages, +5 and -9 V dc, are required. At 25 C, the IC dissipates 1 W. All interconnecting lines are TTL-compatible, and each of the eight data lines can drive one low-power TTL load. Suitable clock-frequencies range from a minimum of about 300 kHz to a maximum of 800.

External circuitry needed

To form a computer system, the microprocessor typically requires the external circuitry shown in Fig. 1. The 8-bit bidirectional data bus connects the microprocessor to the external memory and external registers I_1 and I_2 . During states T_1 and T_2 at phase ϕ_{22} , the control logic strobes data into the external registers. It also determines which way data travel on the bus to and from the microprocessor. In addition the control logic determines when the memory reads or writes, and it activates I/O channels.



1. A computing system using the microprocessor. The external components provide the interface logic for control of the microprocessor and transfer of data to and from memory and I/O devices.

External registers I_1 and I_2 supply addresses to memory as well as data bytes and pointers to the I/O channels. Data from the two most-significant-bit positions of register I_2 are transferred to the cycle decoder. The decoder produces four cycles—PCI, PCR, PCW and PCC—which coordinate the internal operation of the microprocessor with the external circuitry.

When power is applied to the processor chip, 32 clock periods are needed to clear all registers and to initiate the internal microprogram. Then the processor goes into the STOPPED state—as indicated by status bits S_0 through S_2 —until an

Table 1. Nine states execute LMI instruction

Cycle	State	Operation
PCI	T ₁	Low-order bits of program counter to I ₁ . Increment program counter.
	T ₂	High-order bits of program counter and control bits to I ₂ .
	T ₃	Fetch instruction byte from memory location addressed by I ₁ and I ₂ . Put into instruction register.
		Skip states T ₄ and T ₅ and go to the next cycle
PCR	T ₁	Low-order bits of program counter to I ₁ . Increment program counter.
	T ₂	High-order bits of program counter and control bits to I ₂ .
	T ₃	Read byte (immediate data) from memory location addressed by I ₁ and I ₂ . Put into auxiliary register b.
		Skip states T ₄ and T ₅ and go to the next cycle.
PCW	T ₁	Register L out to I ₁ (low-order address)
	T ₂	Register H and control bits to I ₂ (high-order address)
	T ₃	Register b to memory location addressed by I ₁ and I ₂ .
		Skip states T ₄ and T ₅ and go to the next cycle.
END OF EXECUTION OF INSTRUCTION		

interrupt occurs. When this happens—and during states T₁₁ and T₁₂—the address of memory location 0 is referenced for the first instruction byte.

If the first instruction has been loaded into location 0, the instruction byte will be fetched during state T₃. Otherwise the internal instruction decoder attempts to execute the instruction represented by the random-bit pattern in memory location 0. Then the microprocessor sequentially executes instructions or branches, as programmed.

Table 1 shows the execution of typical instruction LMI (Load Memory Immediate—with the next byte in memory following the instruction byte). The instruction occupies two bytes of memory, while its execution requires three cycles. At the beginning of the instruction—and during state T₁—the eight low-order bits of the program counter are sent to register I₁, and the program counter is incremented by one. The six high-order bits of the program counter and the two control bits transfer to register I₂ during state T₂.

At the end of state T₂ the cycle decoder sends a PCI signal to the timing logic, indicating that this is an instruction-fetch cycle. During state T₃ the memory location addressed by the contents of register I₁ and I₂ is read into the instruction register of the microprocessor. The instruction decoder recognizes that this is an immediate type of instruction and transfers control to the appropriate microprogram. Since states T₄ and T₅ are not required, the cycle ends with state T₃.

Similarly the second-cycle and third-cycle operations are executed. Register b, in the second cycle, refers to one of two auxiliary registers (the other is called a). Both are used by the microprogram to transfer data internally. Registers H and L—in the third cycle—are two of seven 8-bit registers internal to the processor chip. The seven are labeled A through E, H and L. They make up the accumulator (register A) and scratch-pad memory (the remaining registers).

Registers H and L should be linked

To execute a memory-reference instruction, the logic takes the 14-bit memory address from the contents of register H (containing the 6 most significant bits) and register L (containing the least significant 8 bits). The instructions contain no field for the memory address.

Since registers H and L can be incremented and decremented, as well as operated logically and arithmetically with register A, it's possible to scan and index all memory locations. When register L is incremented or decremented through a count of zero, register H should be incremented or decremented to continue the scan.

The internal circuitry of the microprocessor does not link registers H and L; so they must be linked by a software subroutine. Table 2 shows one subroutine for incrementing and one for decrementing through all 16k of memory. The contents of registers H and L are independent of the internal instruction address and the program counter.

Table 2. Subroutine links registers H and L

	Label	Instruction	Binary	Comment
INCREMENT REGISTERS	INCHL	INL	00110000	INCREMENT REGISTER L RETURN IF NOT ZERO INCREMENT REGISTER H RETURN
		RFZ	00001011	
		INH	00101000	
		RET	00xxx111	
DECREMENT REGISTERS	DECHL	DCL	00110001	DECREMENT REGISTER L COMPARE AGAINST THE NEXT BYTE ALL ONES RETURN IF NOT MATCHED DECREMENT REGISTER H RETURN
		CPI	00111100	
		377 ^s	11111111	
		RFZ	00001011	
		DCH	00101001	
		RET	00xxx111	

Note: x = don't care

Ready line allows processor to idle

The Ready line of the microprocessor may be activated by a logic ZERO at any time during an instruction. When it is, the microprogram proceeds normally until it reaches the end of a T_2 state. Then, instead of going into state T_3 , the microprogram enters the WAIT state; the microprocessor just marks time by repeating the four clock phases. Deactivation of the Ready line by a logic ONE and after phase ϕ_{22} causes the microprocessor to resume normal operation by going into state T_3 .

The Interrupt line may also be activated at any time during an instruction. When this occurs, the microprogram continues to complete execution of the remaining cycles of the instruction. Then, instead of going to the T_1 state of the PCI cycle, the microprocessor goes to the T_{11} state. As a result, the external circuitry can jam an RST (re-start) instruction onto the data lines during state T_3 . The instruction calls one of the eight locations in low-order memory that contains the subroutine that services the interrupt.

On interrupt the contents of the program counter are not incremented but are pushed down in an internal stack. Hence a RETURN instruction should be programmed at the end of the interrupt service. This instruction causes the program-counter stack to pop up the original program counter, and normal operation is resumed.

Timing vital to processor operation

By means of the cycles and states, the chip conveys information on what is happening internally and what should happen externally. Each instruction requires one, two or three cycles to complete its execution, and each cycle is composed of three, four or five states. In turn, each state is composed of four sequential pulses derived from the system clock.

The clock phases are called ϕ_1 and ϕ_2 (Fig. 2). The chip internally divides ϕ_2 by two to form a signal called SYNC, which is made available to the external circuitry. Each cycle of SYNC contains two pulses from ϕ_1 and two from ϕ_2 —one each from the two phases when SYNC is high and one each when SYNC is low. A complete cycle of SYNC is called a state, and that is made up of four sequential pulses called phases— ϕ_{11} , ϕ_{21} , ϕ_{12} and ϕ_{22} .

Three parallel bits on the status lines define the present state from eight possible states for the chip (Table 3a). Each state begins with the completion of phase ϕ_{22} of the preceding state. The states indicate time slots for functions performed by internal and external operations of the microprocessors.

Normally only three to five of the eight states are used in a cycle. The remaining states can be used for interrupt (T_{11}), direct-memory-access conditions (WAIT) and processor halt (STOPPED).

Under normal operation 8 bits are outputted from the chip onto the external data lines during state T_1 . Since memory addresses require 14 bits, two passes are needed to output the complete memory address. The low-order part of the address is the byte transferred during state T_1 .

The external circuitry does not know at this time whether the contents of register I_1 is part of an I/O instruction, or the memory address of an instruction byte or a data byte.

In the case of an I/O instruction, the byte outputted during state T_1 is the contents of register A rather than part of a memory address. In any event, the byte outputted during state T_1 must be stored in external register I_1 until the external circuitry determines the type of cycle.

During state T_2 the microprocessor delivers to the external data lines the six high-order bits of the memory address (or the pointer to the I/O device) as well as two control bits. The control bits are the two most significant of the byte;

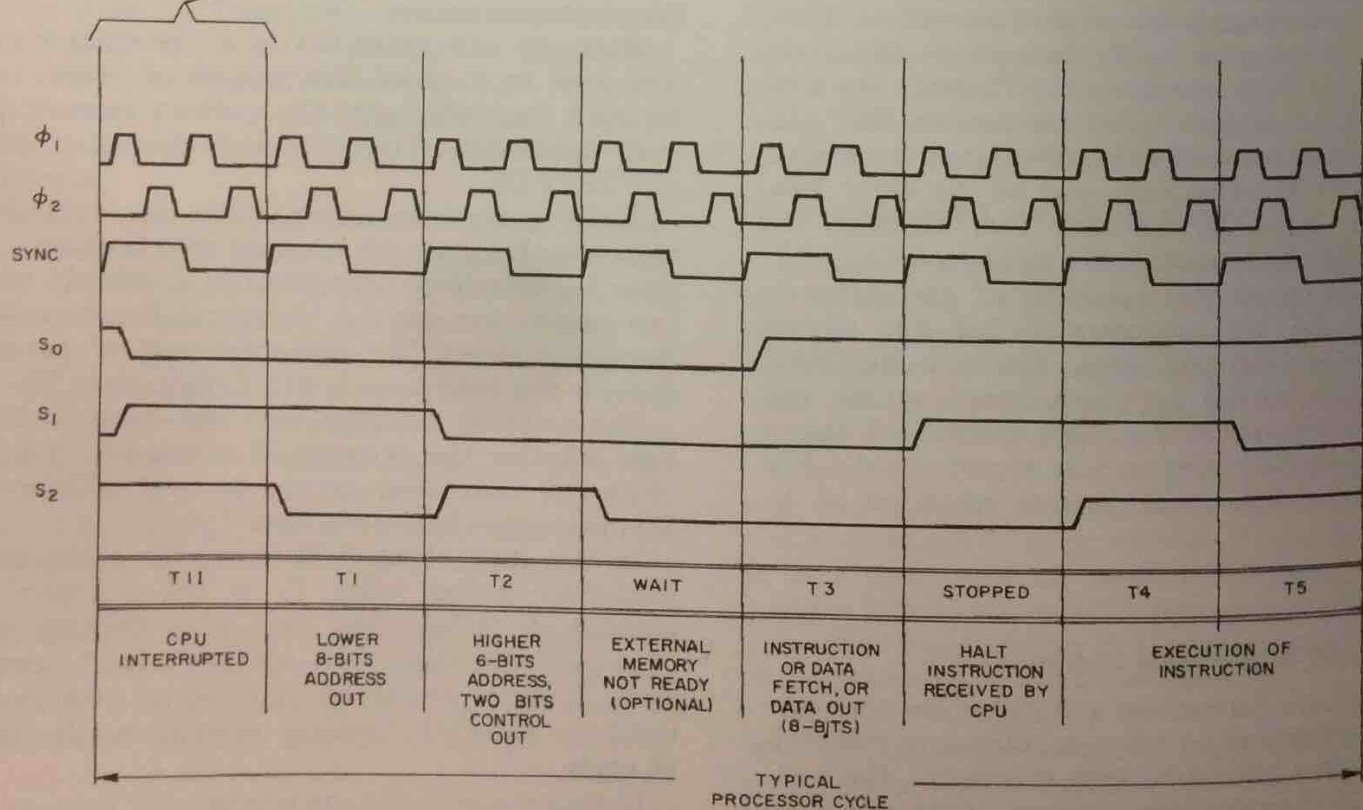
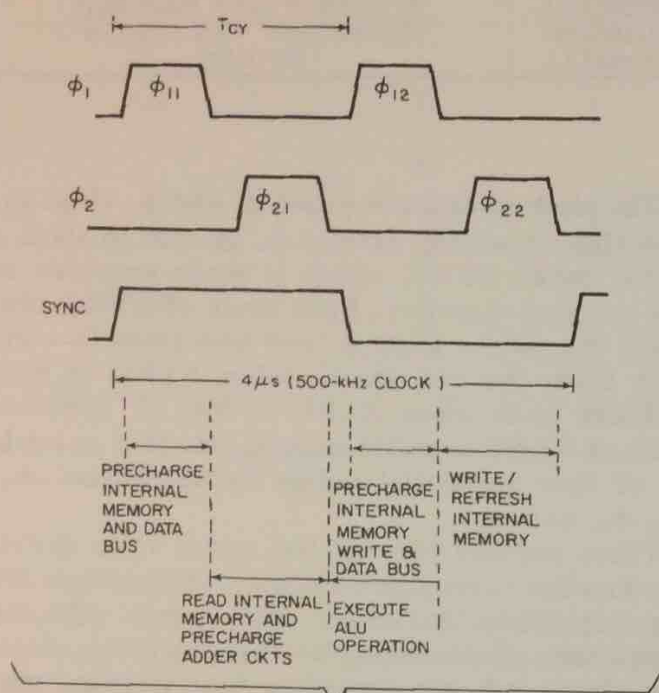
they define which cycle the microprocessor is presently in. The byte outputted during state T_2 must also be stored in an external register, I_2 , until the cycle has been decoded (Table 3b). The cycle defined by the two control bits determines what is to be done with the contents of the two registers.

In state T_3 the microprocessor either inputs a

byte of data, fetches an instruction byte or outputs a byte of data. These transfers are performed via the bidirectional data bus, with the particular operation depending on the instruction being executed.

During states T_4 and T_5 the microprocessor executes the instruction and transfers data between its internal registers. Some instructions do not require these states. In these cases the states are either left idle or the cycle ends by skipping to state T_1 .

Whenever a HALT instruction occurs, the microprocessor goes into the STOPPED state. The internal registers continue to refresh themselves periodically, maintaining the stored data. The microprocessor remains STOPPED until it receives an interrupt signal. It then goes to the T_{11} state at the beginning of the next instruction. When this is detected on the status lines, the INTERRUPT line should be put back to logic ZERO; the microprocessor then resumes normal operation.



2. Internal microprogram control uses cycles and states to convey relevant internal and expected external operations to the rest of the circuitry. A typical execution

time for nonmemory instructions is 20 μ s (with a 500-kHz clock), which covers states T_1 , T_2 , T_3 , T_4 and T_5 —or a typical processor cycle.

Two control bits identify cycles

A specific microprogram cycle is defined by the two control bits (most significant two bits) of external register I_2 at the end of state T_2 . Each cycle performs a particular portion of an instruction, which may require one, two or three cycles for completion.

Each instruction must begin with a PCI cycle, which fetches the next instruction. The instruction's location in memory is contained in the program counter and transferred to registers I_1 and I_2 during states T_1 and T_2 , respectively. Fig. 3 shows the processor-state transitions for each cycle.

During the fourth phase, ϕ_{22} , of state T_2 it is convenient to strobe the two control bits through combinatorial logic into one of four cycle flip-flops. In this manner the external circuitry knows which cycle is being performed.

For the PCI cycle the contents of the addressed memory location are put onto the data bus during the next sequential state (T_3). The data byte enters the microprocessor and transfers via the internal data bus to the instruction register and decoder, which determines the operation to be performed. The program counter is incremented by one during the PCI cycle.

Some instructions—such as Register-Register, Register-Arithmetic-Logic, Rotate and Return—are executed internally during states T_4 and T_5 of the PCI cycle. For other instructions, the PCI cycle terminates with state T_3 , and additional cycles are required to complete the execution.

When either the PCR (memory-read) or PCW (memory-write) cycles are indicated, the external register, I_1 , holds the contents of register L. The six least significant bits of external register I_2 hold the contents of register H.

For a PCR cycle, the contents of the addressed memory location appear on the data bus during state T_3 . The data byte is entered into the microprocessor, and transferred via the internal data bus to the designated register. Depending upon the particular instruction, states T_4 and T_5 may or may not be used.

For a PCW cycle, the contents of the appropriate register, as designated by the instruction, will appear on the data bus. The data byte is written into the memory location addressed by the contents of registers I_1 and I_2 . States T_4 and T_5 are skipped during a PCW cycle, and the cycle terminates with state T_3 .

The PCC cycle is used only for INPUT and OUTPUT instructions. These instructions are composed only of a three-state PCI cycle followed by a PCC cycle.

At the end of state T_1 , external register I_1 holds the contents of register A (accumulator). External register I_2 holds the least significant

Table 3. Status and control bits yield states and cycles

State	Status Bits		
	S_0	S_1	S_2
T_1	0	1	0
T_2	0	0	1
T_3	1	0	0
T_4	1	1	1
T_5	1	0	1
T_{11}	0	1	1
WAIT	0	0	0
STOPPED	1	1	0

3a

Cycle	Control Bits		Function
	MSB	2SB	
PCI	0	0	INSTRUCTION FETCH CYCLE
PCR	1	0	MEMORY READ CYCLE
PCW	1	1	MEMORY WRITE CYCLE
PCC	0	1	I/O CYCLE

3b

six bits of the instruction byte that was transferred during state T_2 . The instruction byte contains a field of bits that points to the particular I/O device addressed. The instruction also indicates if it is an input or an output operation.

If an output operation is called, state T_3 is idle and the microprocessor merely marks time. Meanwhile the external circuitry must transfer the contents of register I_1 (contents of the accumulator) to one of 24 possible output devices, as indicated by the pointer field in register I_2 . Register I_1 contains the data byte placed into register A before the OUTPUT instruction was executed. The cycle ends with state T_3 .

To call an input operation, the input byte must appear on the data bus prior to state T_3 . During state T_4 conditional flags will appear on the data bus. These may be examined for test purposes or to show status conditions. At the end of state T_5 the inputted data byte appears in register A.

The input device may be addressed in one of two ways, depending on the external circuitry. One technique uses the pointer field in register I_2 —as was done during the output operation. This allows only one of eight possible input devices to be addressed. A different instruction is required to input from each device, because of the pointer field in the instruction byte.

The second method uses the entire 8-bit byte in register I_1 (instead of the pointer field of register I_2) to address one of 256 possible input devices. Since the contents of register A are placed into external register I_1 during T_2 , this technique requires the device address to be placed

into register A before execution of the IN-PUT instruction. Thus a single input instruction can address several input devices by changing the contents of register A.

230 instruction codes are possible

Most of the microprocessor's basic set of 48 instructions have modifiers, such as registers or condition flags, that result in a total of 230 individual instruction bytes.

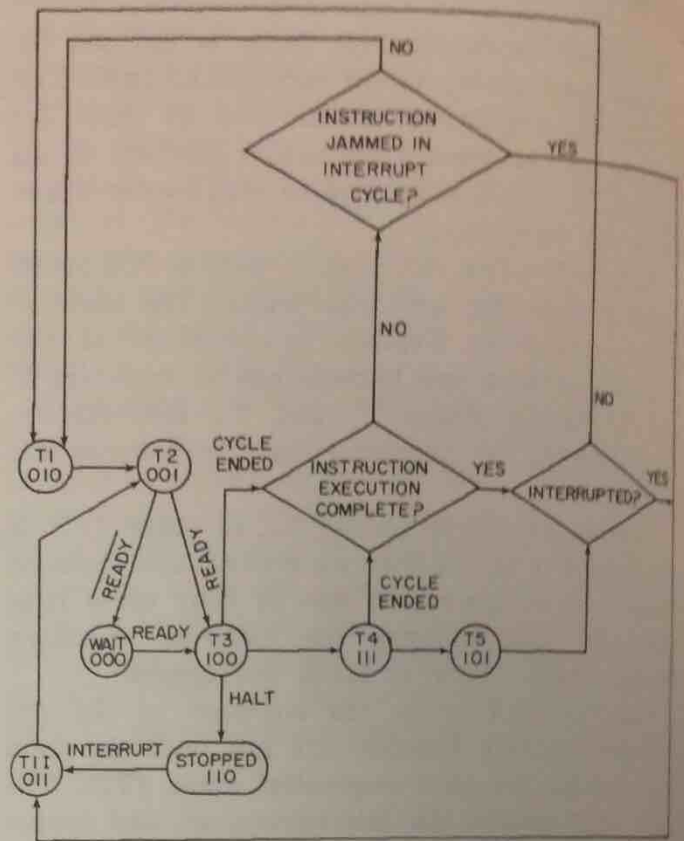
Instructions that perform internal operations are one byte long. Immediate type of instructions are two bytes in length, since the data is in the byte after the instruction. Three-byte instructions are used for call and jump operations, since a two-byte address follows the instruction byte.

There are six types of basic instructions. The register operations include register-register, register-memory and increment or decrement. The arithmetic operations are addition with or without carry or subtraction with or without borrow. The logic operations include AND, OR, EXCLUSIVE-OR and COMPARE, as well as ROTATE register A left or right. Most instruction operations can have as modifiers any of the seven registers, the contents of memory (addressed by the contents of registers H and L), or immediate data (next byte). The exceptions are INCREMENT, DECREMENT and ROTATE register A.

The transfer operations cover JUMP, CALL and RETURN. These may be unconditionally executed or executed on one of four conditional flags either true or false. The flags are CARRY, ZERO, SIGN and PARITY.

Using the COMPARE instruction, tests can be made for less-than, greater-than or equal conditions. The ZERO flag, when set after a COMPARE instruction, indicates an equality. If not set, the CARRY flag determines less-than or greater-than conditions.

An additional transfer instruction, RST, is a one-byte call. Normally a call requires three bytes to give the two-byte branch address following the



3. Each cycle causes microprocessor state transitions. A cycle begins when status lines indicate state T_1 .

instruction byte. But the RST instruction has embedded in its bit pattern the location of the branch. The microprocessor has eight individual low-order memory locations that the RST instruction can call.

The bit pattern of the RST instruction is derived by adding the number 5 to the address of one of the eight individual memory locations. An example of this: To call location address 0, the RST instruction would be 5, or a bit pattern of 00000101.

Bibliography:

MCS-8 Micro Computer Set, Intel Users Manual, March, 1973, Revision 3.

Clearing the Interface and Software Hurdles

DONALD R. LEWIS and W. RALPH SIENA
Automata Systems Corp., Kew Gardens, N.Y.

The foundations of successful microprocessor use are the choice of the right interface and efficient software development.

As with any computing system, the peripheral devices for use with microprocessors must meet system requirements. They require an interface to the microprocessor. The right choice ensures that the advantage of minimal hardware—gained by the use of microprocessors—will not be lost because of an overdesign of the interface.

Similarly software can be the largest single cost in any computer. With microprocessors, the ratio of software-to-hardware costs can easily exceed that of conventional computers. Hence there is need for an efficient procedure to develop software.

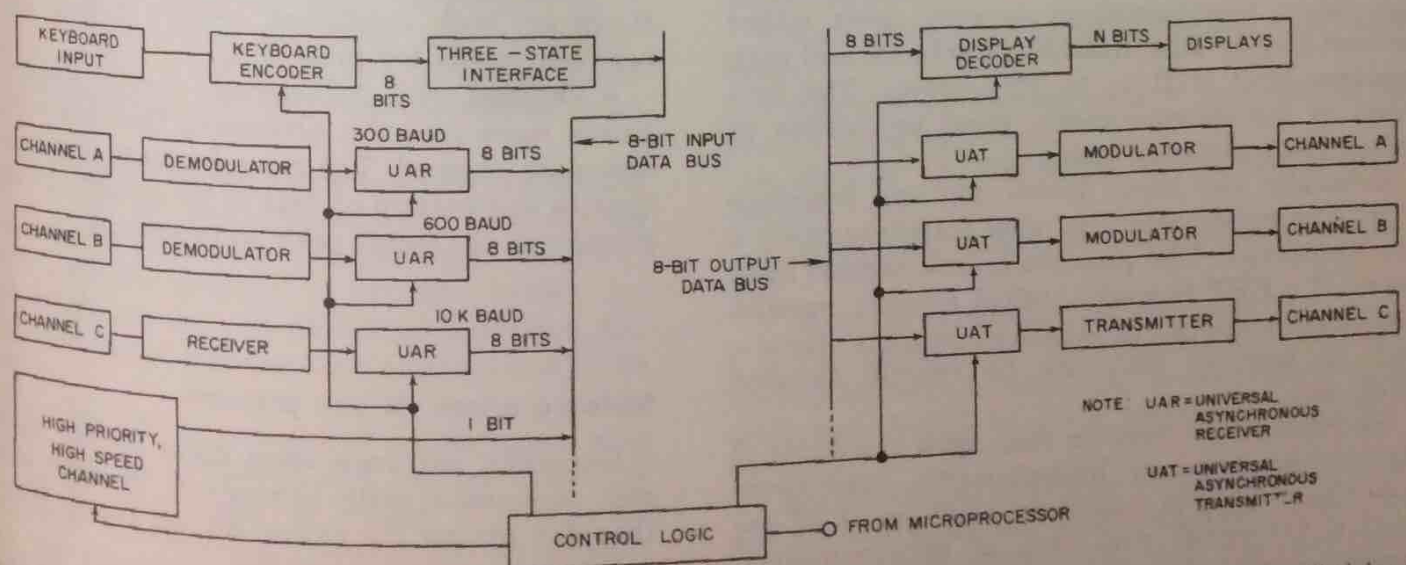
Microprocessors can operate with data links that are either direct or telecommunication types. Either type permits data to be transmitted serially or in parallel on a bus.

A telecommunication link, unlike a direct link, requires a modulator/demodulator, or modem, to

interface the system to the link. Modems supply the carrier frequency and decode transmitted information. Modem designs can be simplified by the use of universal asynchronous receivers and transmitters, or UARTs (Fig. 1). These are available as single MOS/LSI chips, featuring MOS or DTL/TTL-compatible inputs and outputs. For serial transmission, baud rates can range from dc up to 40 kHz.

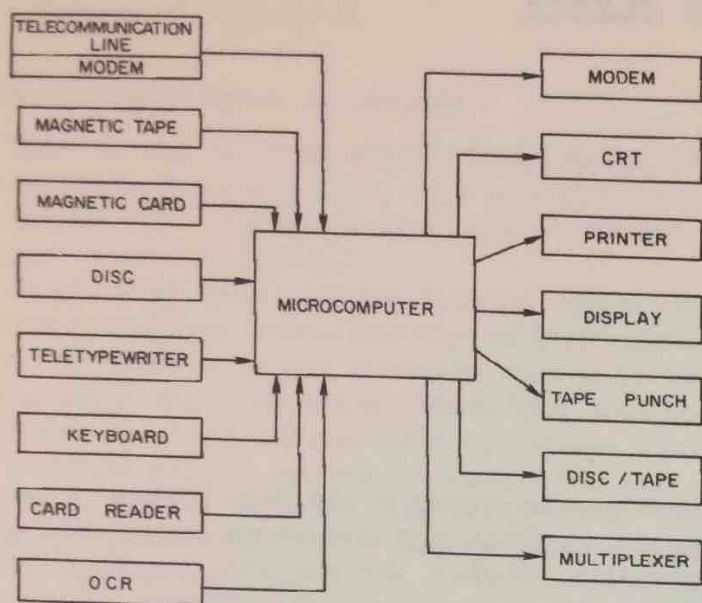
Baud rates on the input channels also affect both the design of the interface and the handling of channels. A telecommunication channel can operate with baud rates up to 2400 without conditioned telephone lines. But higher rates of 4800 and 9600 baud need conditioned lines.

A single microprocessor can handle several channels, each operating at a different baud rate. For example, systems can be built with telecommunication channels operating at 300 and 600 baud rates and with data links handling 10,000 baud. And the microprocessor provides simultaneous, asynchronous control.



1. The determining factors in the interface design are word length, speed and priority. Transmission of dif-

ferent baud rates and priorities can be handled by the I/O channels of a single microprocessor.



2. Any computer peripheral can work with microprocessors. Interface requirements are determined by word length and any additional hardware needed.

Keyboard inputs, both numeric and alphanumeric types, can also be used with microprocessors. Standard codes—such as ASCII, EBCDIC and BAUDOT—as well as special codes may be employed, depending on system requirements. When the microprocessor is interfaced with a keyboard, the code can be written in the instruction format. For example, all 48 instructions of an 8-bit parallel microprocessor can be derived from a single keyboard.

Microprocessors permit direct operation from teletypewriters. The prime consideration is one of level shifting to obtain a TTL-compatible input. Most teletypewriters operate from either current-loop or EIA RS232C interfaces, and each requires level shifting to TTL levels. The program of the microprocessor must be written to accept teletypewriter codes.

Generally any input device that can interface with a minicomputer can also interface with a microprocessor. The possible inputs include light pens—if a CRT is used—optical readers, sensors and card readers. In addition storage inputs, such as tape or disc, can be readily handled (Fig. 2).

Microprocessor outputs may pose problems if the system requires high-capacity, high-speed data channels. The relatively slow speed of operation, compared with that of minicomputers, seriously limits the number of such channels that can be processed by a single microprocessor.

However, all types of output devices can still be used.

Interfaces: Check format compatibility

In the design of the interface, format compatibility is important. This includes both word length and field arrangement. The right selection can minimize data-transmission time and circuit complexity, and simplify the necessary software.

The word length affects the amount of hardware required at the interface. A fixed word length can be standardized for a number of I/O devices. It is much easier to implement and control than a variable word length, which differs for each device.

A field-oriented data word can offer further improvements. Words with 4-bit fields or words with one 2-bit and two 3-bit fields can be used within the same system and sometimes even within the same channels. Use of field-oriented words results in decreased transmission times and increased efficiencies.

Another major consideration—channel priorities—depends on the data and traffic rates. Many channels have varying levels of importance to the system. Some channels may have to be processed faster or more often than others. This would call for higher data rates.

Where many inputs and outputs are serviced, varying data rates are obviously preferable to a processor overload. However, this approach reduces the data that can enter the system at any one time, as well as the time for processing of the data. For example, an 8-bit serial data word with a start bit, a stop bit and a parity bit—a total of 11 bits—operating at a 300-baud rate, requires 36.6 ms to transmit one word. A channel with the same data at 600 baud requires only 18.3 ms. Hence twice as much data can be transmitted in the same time frame in the second case. However, both conditions require a long time between words for processing.

A channel with considerable traffic often requires more frequent processing than a low-volume data channel. The more frequent processing implies less backup of data at the output and less storage at the input prior to processing.

Software brings its own problems

Generally software costs make up the major cost component with computers. In some cases they far exceed the outlay for hardware. Microprocessors are no exception.

The software phase of the design (Fig. 3) calls for performance of the following major tasks:

- System definition.
- Equating definition to programs.
- Program design.
- Charting of functional flow and detailed

- flow.
- Instruction writing, or coding.
- Debugging.
- Editing.
- Final program layout, or ROM stacking.

Program design tends to be more detailed than that for a conventional computer. The basic assumption here is that the microprocessor functions in a stand-alone application and that, as a result, ROMs will be used to store the entire program. This restriction limits the total program unless bank-switching techniques are used. These increase the apparent size of the memory beyond the maximum rating for a microprocessor.

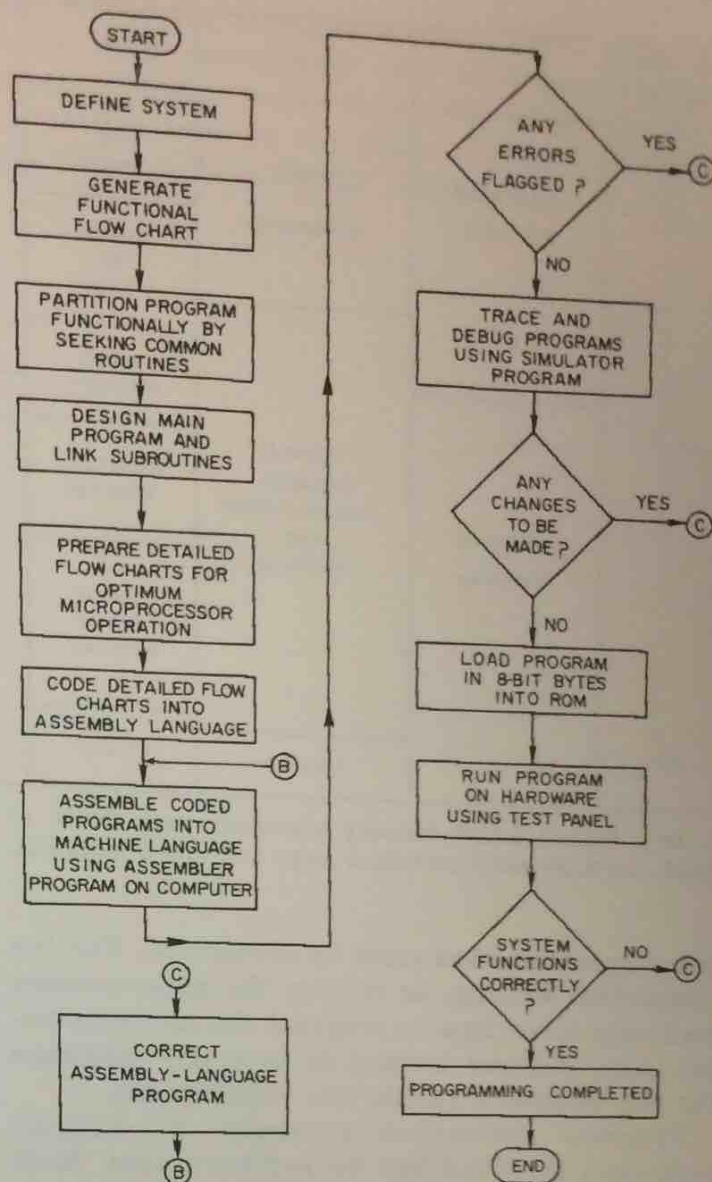
System definition involves the major tasks to be performed by the microprocessor—which is assumed to be the central control device of the system. Data formats should be established to maximize processor control. The over-all system timing is included in this design phase to ensure that all functions can be handled within the timing constraints.

Based on the system definition, the basic program structure can now be defined. Each input channel to the microprocessor represents a major program, assuming the use of more than one input device. In addition an Executive program should be written to control the over-all operation of the system. Various routines—based on the different functions or command codes supplied—further subdivide the main program.

Since most microprocessors are designed around an 8-bit bus, the program can be made more efficient if it operates with a maximum of 8 bits per message. This eliminates the need for multiple-word processing for data control. And the control of command codes as a function of input devices allows flexibility and efficiency. The same data word can have a different meaning, or subroutine, for each input device or channel.

The program design, the single most important part of the software development, bridges the gap between hardware and software—or firmware when the program is put into ROMs. Involved here are the over-all system operation, the hardware design and the kind of programs to be written.

The program design should define every step of the system operation from the point-of-view of the microprocessor. It should establish the necessary "handshaking" between the peripherals and the microprocessor and between the peripherals and external circuitry. Some of the specifics that the program design should include are:



3. The software development can be achieved efficiently with a flow chart. As part of the final design steps, a test panel, or control unit, should be built to test the program after it has been loaded into ROMs.

as follows:

- System-timing requirements.
- Code structures between the system and external circuitry.
- Code structures to and from the microprocessor.
- Sources of microprocessor data during all phases of its operation.
- Method of Executive and other program control.
- Method of program interface and interaction.
- Memory structure and operation.

A functional flow chart defines the functional operations sequentially. But it does not contain sufficient detail to allow program writing or coding.

Detailed charts overcome this limitation. They are derived from the functional flow charts, the established command codes and the manner in which each command code works within the system. The detailed flow charts tell, step by step,

PAGE LINE	0	1	2	3	4	5	6	7	
0	INTERRUPT ROUTINES	TABLE OF CONSTANTS	BCD TO BINARY	UTILITY ROUTINES	PROCESS CONTROL FOR EXTERNAL HARDWARE	LOG CONVERSION	CRT DISPLAY DRIVER	BUFFER AREA FOR RAM	
1			BINARY TO BCD						
2		EXECUTIVE PROGRAM	DOUBLE PRECISION MULTIPLICATION AND DIVISION			INPUT ROUTINE			RESERVED FOR FUTURE EXPANSION
3						PRINTOUT ROUTINE			
4	ERROR ALARM ROUTINE								
5			UNUSED						
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									
16									
17									
18									
19									
20									
21									
22									
23									
24									
25									
26									
27									
28									
29									
30									
31									
32									
33									
34									
35									
36									
37									
38									
39									
40									
41									
42									
43									
44									
45									
46									
47									
48									
49									
50									
51									
52									
53									
54									
55									
56									
57									
58									
59									
60									
61									
62									
63									
64									
65									
66									
67									
68									
69									
70									
71									
72									
73									
74									
75									
76									
77									
78									
79									
80									
81									
82									
83									
84									
85									
86									
87									
88									
89									
90									
91									
92									
93									
94									
95									
96									
97									
98									
99									
100									
101									
102									
103									
104									
105									
106									
107									
108									
109									
110									
111									
112									
113									
114									
115									
116									
117									
118									
119									
120									
121									
122									
123									
124									
125									
126									
127									
128									
129									
130									
131									
132									
133									
134									
135									
136									
137									
138									
139									
140									
141									
142									
143									
144									
145									
146									
147									
148									
149									
150									
151									
152									
153									
154									
155									
156									
157									
158									
159									
160									
161									
162									
163									
164									
165									
166									
167									
168									
169									
170									
171									
172									
173									
174									
175									
176									
177									
178									
179									
180									
181									
182									
183									
184									
185									
186									
187									
188									
189									
190									
191									
192									
193									
194									
195									
196									
197									
198									
199									
200									
201									
202									
203									
204									
205									
206									
207									
208									
209									
210									
211									
212									
213									
214									
215									
216									
217									
218									
219									
220									
221									
222									
223									
224									
225									
226									
227									
228									
229									
230									
231									
232									
233									
234									
235									
236									
237									
238									
239									
240									
241									
242									
243									
244									
245									
246									
247									
248									
249									
250									
251									
252									
253									
254									
255									

for each address are listed sequentially, 000 through 255.

For maximum flexibility, pROMs should be programmed by the designer. The aids available allow both initial program writing into the pROM and copying from a completed pROM to a new

one. Where the program instruction set takes up more than one-quarter of the total memory space, in-house programming has minimal effect on testing time. At this phase of design, the testing time can be critically short.

SECTION III

Grappling with Microprocessor Limitations

In general, techniques described in this section enhance specific microprocessors. For example, the first article shows how to make a rather fast microprocessor even faster. Other articles cover microprogramming techniques and external circuitry that circumvents microprocessor limitations.

Also included in this section is an article that explains how to use mini-computers to debug microprocessor systems. The technique should prove especially helpful for micros that have small memory and limited peripheral equipment and, therefore, cannot accommodate diagnostic software.

Speeding Microprocessor Multiplication	81
<i>Hermann Schmid, Senior Engineer-Computers, General Electric Co., Binghamton</i>	
Microprogramming Extends LSI-Processor Capabilities	89
<i>George Reyling, Jr., Project Manager, National Semiconductor Corp., Santa Clara</i>	
Debug that Microcomputer System with a Mini	95
<i>Raic Dusan, Research Engineer, ISKRA, Ljubljana, Yugoslavia</i>	
Improving Microprocessor Interrupt-Handling Capability	99
<i>Tom Pittman, Microprocessor Consultant, San Rafael, CA</i>	
Providing the Clock and Drive Signal for the 8080	101
<i>Gundars Osvalds, Senior Technician, Communication Satellite Corp. Laboratories, Clarksburg, MD</i>	

Speeding Microprocessor Multiplication

HERMANN SCHMID
Senior Engineer-Computers,
General Electric Co., Binghamton

When required to multiply, microcomputers—LSI microprocessors plus support circuits and memories—may be too slow for a host of real-time control applications. Typically these applications require multiplication times of 5 to 50 μ s. Microcomputers need several orders of magnitude longer. But with external circuitry—either a circuit that complements the CPU or a separate, peripheral multiplier—the speed limitations can be overcome.

The complementary-circuit approach requires a microprocessor that can be microprogrammed and has an externally accessible control bus. The circuitry differs from one processor to another. However, the approach yields the highest speed and least hardware complexity.

A separate peripheral multiplier can be used with any LSI processor. But it is less efficient from the standpoints of time and hardware: An 8 \times 8-bit multiplier requires nine MSI circuits and at least four clock periods; for a 16 \times 16-bit multiplier, these requirements are doubled.

A complementary circuit for National Semiconductor's IMP-16C microcomputer¹ can be built with 16 standard SSI and MSI circuits. And these can be interconnected on a 3 \times 4-in. PC board that is mounted piggyback on the microcomputer.

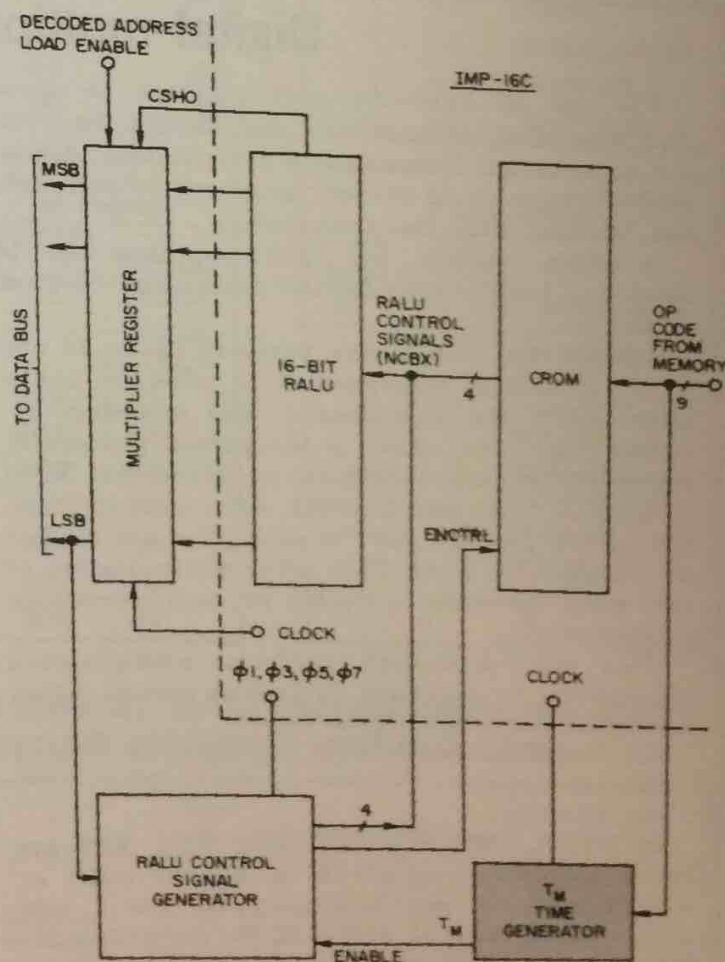
The complementary circuit has been designed around the National unit, because it was the first available 16-bit model. Other models have been announced.

With the complementary circuit, the National microcomputer allows multiplication of two 16-bit unsigned operands in 16 microcycles, or 23 μ s, with a 6.5-MHz clock. Thus the multiplication time is reduced by a factor of 30 from the relatively fast 700 μ s needed by a conventional macro-software operation. It is reduced by a factor of seven from the 150 μ s needed by an optional microprogrammed instruction offered by National.

Speed benefits also result when a complementary approach is used for division or square-root operations, or for multiplication of signed oper-

ands. The same hardware technique is used for these operations, but with some increase in ICs.

The design of the complementary circuit for multiplication assumes these three requirements: (1) The basic operation of the microcomputer will not be disturbed or impeded; (2) The additional circuitry will provide the necessary bipolar or MOS interface levels, and (3) No addi-



1. In this CPU complementary multiplication circuit, CROM outputs are replaced with special control signals that permit hardware multiplication operations. The additional circuit blocks consist of a time generator, signal generator and multiplier register.

tional power supplies will be used.

The major blocks of the multiplication circuitry are connected to the RALU (register and arithmetic logic unit) and CROM (control read-only memory) of the IMP-16C (Fig. 1). The external circuitry consists of the T_M time generator, the RALU control-signal generator and the multiplier register (MR).

Key microcomputer operations

In a typical microcomputer operation, the CROM receives a 9-bit operational (op) code from memory and processes it into the RALU control signals (NCBX). This time-sequenced 16-bit control word instructs the RALU what to do at each phase of a microcycle. At each clock phase, the four lines determine the following:

- During phase ϕ_1 , which register (or stack) is connected to the "A" bus. The "A" and "B" buses constitute the two ALU input buses.

- During ϕ_3 , which register is connected to the B bus and also whether to complement the A bus.
- During ϕ_5 , which arithmetic logic and control operations are to be performed.

- During ϕ_7 , which signal bus is to be connected to the "R" bus (the ALU output bus) and into which register (or stack) the R bus is to be loaded.

For multiplication, the RALU control signals from the CROM are replaced with separately generated control signals. These are a function of the least-significant MR register bit. The switchover in control signals occurs upon detection of a special multiplication op code—not in the instruction repertoire. The op code simply turns the CROM off and the T_M time generator on.

The time generator provides a period 16 microcycles long for a 16-bit multiplier, and it starts one microcycle after receipt of the multiplication op code. Also, T_M connects the shift clock to the MR register. A function of this register is to

Digital multiplication: The basics

The use of hardware multiplication to speed microcomputer computations also entails the writing of software. Though not a difficult task, digital multiplication could present problems to designers not familiar with the procedure.

A simple example (top right) will show how to develop a basic multiplication subroutine (bottom right).

Assume that two binary numbers, $X = 13$ and $Y = 11$, must be multiplied. X is called the multiplier and Y the multiplicand. The procedure requires that Y be added to the partial product Z_i whenever the least-significant multiplier bit X_i is 1. When $X_i = 0$, zero is added. After each addition, the partial product and the multiplier are shifted to the right by 1 bit. Thus after multiplication of two n -bit operands, a $2n$ -bit or double-precision product results.

Most microprocessors perform multiplication sequentially by software. A typical multiplication subroutine consists of three steps:

- (1) Initialize,
- (2) Loop and
- (3) Finalize.

In Step 1, CPU registers AC0, AC2, AC3 are loaded with the operands X , Y and the index n ; AC1 is reset to zero. In Step 2, we add Y into AC0 when the LSB of AC0 $\neq 0$. We omit the addition when the LSB of AC0 = 0. The contents of AC0 and AC1 are shifted to the right 1 bit at a time, for each pass through the loop, while the index counter is decremented. In the last step we transfer the double-precision product from AC0 and AC1 to specified memory locations.

$$Y = 11$$

$$\begin{array}{r} 1011 \end{array}$$

$$X = 13$$

$$\begin{array}{r} 1101 \end{array}$$

x

$$Z_0 = \begin{array}{r} 0000 \\ +1011 \\ \hline \end{array}$$

$$Z_1 = \begin{array}{r} 1011 \\ 1011 \longrightarrow \text{SHIFT RIGHT} \\ +00000 \\ \hline \end{array}$$

$$Z_2 = \begin{array}{r} 01011 \\ 1011 \longrightarrow \text{SHIFT RIGHT} \\ +101100 \\ \hline \end{array}$$

$$Z_3 = \begin{array}{r} 110111 \\ 110111 \longrightarrow \text{SHIFT RIGHT} \\ +1011000 \\ \hline \end{array}$$

$$Z_4 = 1000111$$

143 = DOUBLE-PRECISION ANSWER

OPERATIONS: AC1, AC0 \leftarrow AC0 x AC2

INITIALIZE: AC0 \leftarrow X, AC1 \leftarrow 0, AC2 \leftarrow Y,
AC3 \leftarrow n

LP: JUMP +2 IF AC0 LSB = 0

ADD AC1 \leftarrow AC1 + AC2

RIGHT SHIFT AC1 LSB \rightarrow L

RIGHT SHIFT AC0 L \rightarrow MSB

DECREMENT COUNTER, SKIP IF ZERO

JUMP TO LP

FINALIZE: STORE AC0 AND AC1 IN MEMORY

initially hold the 16-bit multiplier.

At each clock cycle, or microcycle, the multiplier shifts one bit to the right. And as the least-significant multiplier bit (MR-LSB) leaves the low side of the register, the least-significant product bit enters the top side. At the end of T_M , the MR register thus holds the 16 low-order product bits.

Use of an external register to store the multiplier operand eliminates the following:

- The need to shift through the "Link" flip-flop (CPU status flag), which would require two microinstructions.
- Testing of the multiplier LSB through software, which would require a conditional branch microinstruction.
- The need to establish, increment and test the index counter, which would require another two microinstructions.

The RALU control signals (NCBX) are generated with simple logic circuits and connected to the NCBX bus during T_M (Fig. 2). Only one transistor, four Tristate MOS buffers and six diodes are needed.

The multiplicand is loaded into accumulator AC2, and the multiplier into the external register. During each microcycle either the content of AC2 (MR-LSB = 1) or zero (MR-LSB = 0) is added to the content of AC0, which is initially zero. In addition the contents of the AC0 and MR registers shift 1 bit right, and the content of the least-significant AC0 stage shifts into the most-significant MR register stage. At the end of the multiplication operation, the most-significant product byte is in AC0 and the least-significant byte in the MR register.

For this design, a macroinstruction loads the multiplier operand into the MR register prior to the multiplication operation. Similarly another macroinstruction causes the 16 low-order product bits to be read out from the register after multiplication.

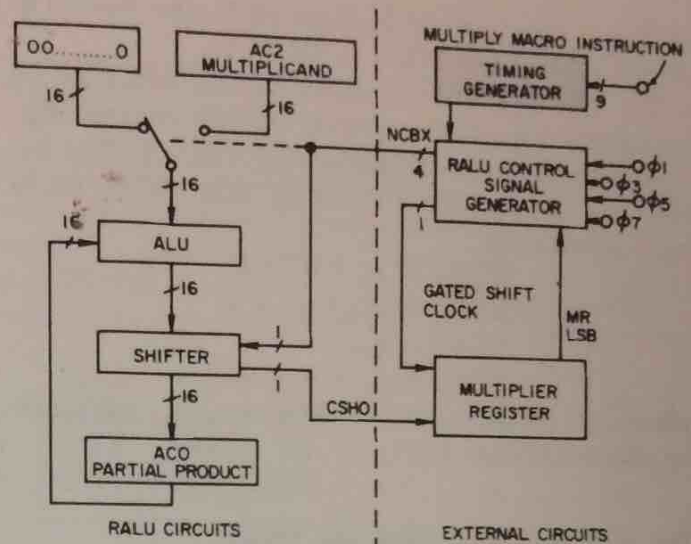
Obviously the T_M period and the RALU control signals can be extended, so these two operations are performed at the high speed of 1 microcycle each without additional software. But for simplicity, this approach is not used.

Control signal pattern easy to generate

To perform a hardware multiplication operation, two pseudo-microinstructions must be generated and then executed. These are the following:

- (AC0) \leftarrow [(AC0) + 0] 2^{-1} if MR-LSB = 0 (1)
 (AC0) \leftarrow [(AC0) + (AC2)] 2^{-1} if MR-LSB = 1 (2)

Translated, this means: (1) Add zero to the content of AC0 and shift the result 1 bit right if MR-LSB = 0; (2) Add the content of AC2



2. The complementary circuit provides alternative control signals, labeled NCBX, for the RALU. The technique can be used with the IMP-16C because it is a microprogrammable processor.

Table 1. Command codes for the RALU

ALU functions		Control functions	
NCB (1), (0) @ T5	Function	NCB (3), (2) @ T5	Function
11	AND	11	None
10	XOR	10	R-bus control
01	OR	01	Shift left
00	ADD	00	Shift right

A, B and R-bus addresses		R-bus control	
NCB (2) (1), (0)	Address	I/O NCB(3) @ T7	BYTE (SININ @ T5)
111	ZEROS		
	FLAGS, STACK		
110	R1	1	0
101	R2	1	1
100	R3	0	0
011	R4	0	1
010	R5		
001	R6		
000	R7		

R-bus value	
I/O NCB(3) @ T7	BYTE (SININ @ T5)
1	0
1	1
0	0
0	1

Table 2. RALU signal patterns that permit unsigned multiplication

	Time period	NCB3	NCB2	NCB1	NCB0	Operation
Multiplier MR-LSB = 0	ϕ_1	1	1	1	1	(A bus) \leftarrow 0
	ϕ_2	1	0	1	1	(B bus) \leftarrow AC0
	ϕ_3	0	0	0	0	SHIFT RIGHT, ADD
	ϕ_4	1	0	1	1	(AC0) \leftarrow (R bus)
Multiplier MR-LSB = 1	ϕ_1	1	0	0	1	(A bus) \leftarrow (AC2)
	ϕ_2	1	0	1	1	(B bus) \leftarrow (AC0)
	ϕ_3	0	0	0	0	SHIFT RIGHT, ADD
	ϕ_4	1	0	1	1	(AC0) \leftarrow (R bus)

to the content of AC0, then shift the result in AC0 1 bit right if $MR - LSB = 1$.

The two pseudo-microinstructions can now be translated into the required RALU control signals with the code definitions in Table 1. The result is a truth table (Table 2) that yields these logic equations:

$$NCB0 = \phi_1 + \phi_3 + \phi_7$$

$$NCB1 = \phi_{18} + \phi_3 + \phi_7$$

$$NCB2 = \phi_{18}$$

$$NCB3 = \phi_1 + \phi_3 + \phi_7$$

where ϕ_{18} denotes the clock phase, ϕ_1 , switched by the multiplier LSB.

The problem: Interfacing and timing

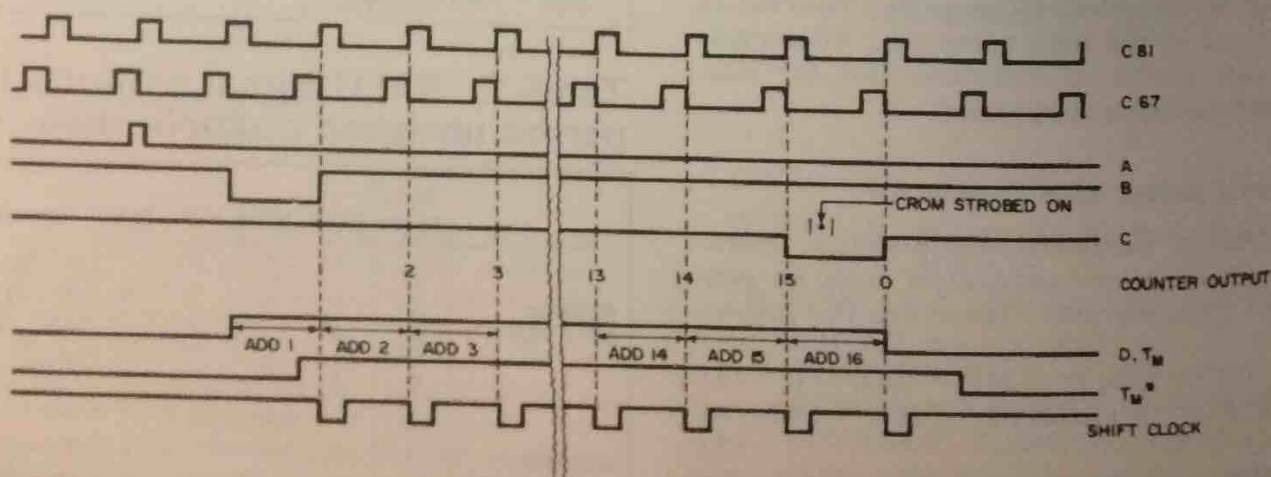
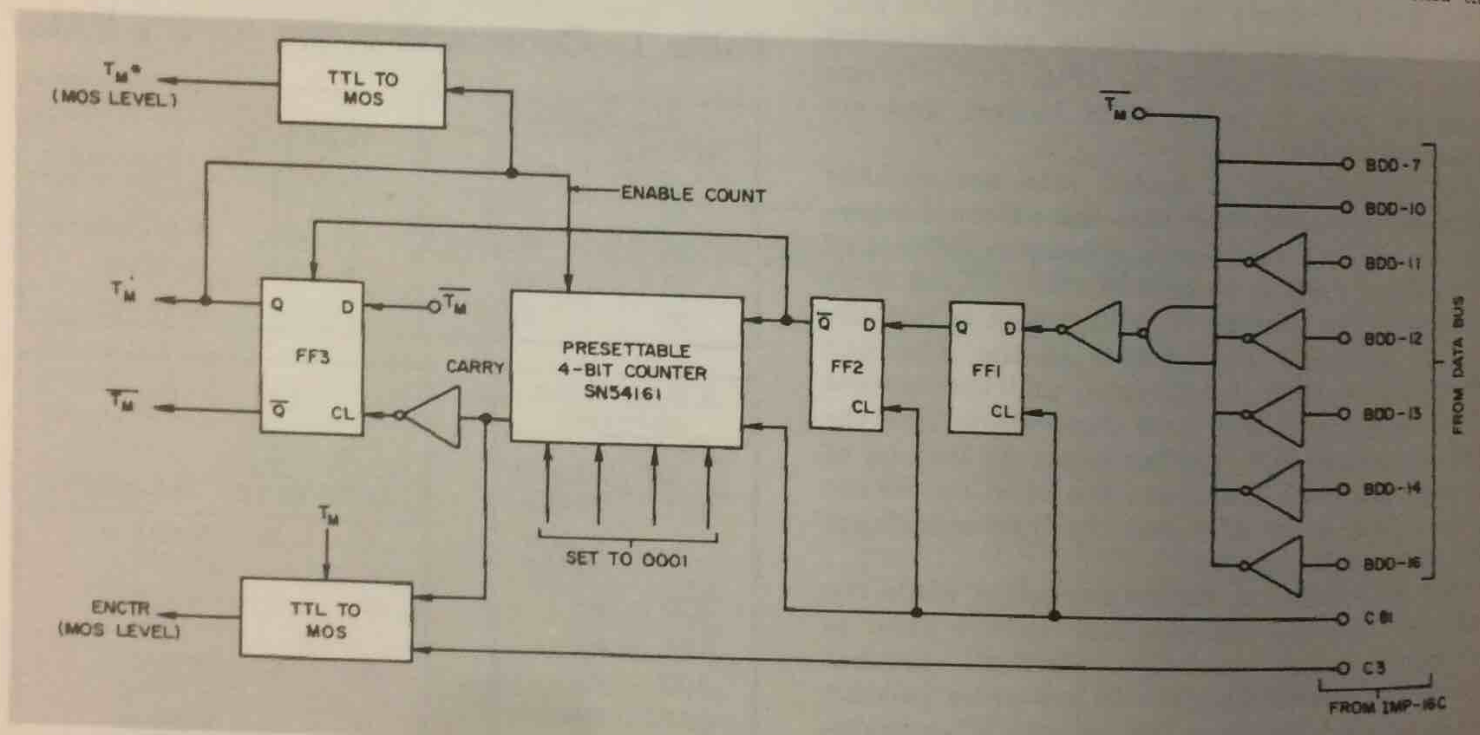
Implementing these logic equations in either TTL or MOS levels would be a cinch. However, the RALU and CROM have both types of inputs

and outputs, even though the chips use MOS techniques. The outputs may be either the standard pull-up/pull-down types, open collector or Tri-state.

As shown in Table 3, some of the RALU control signals perform different functions within the eight time periods of 1 microcycle. For example, CSHO (carry-shift-zero) accepts a carry input during period T_3 and outputs the shift pulse during T_8 . Also, the NCBX signals are always driven to logic ZERO during even clock phases.

The timing of these signals is very critical. For example, the carry output signal (CSHO) is guaranteed to be available only during the last 70 ns of T_8 . Similarly the pseudo-NCBX signal cannot be delayed by more than 85 ns from the start of the clock phase.

Consequently logic levels, impedances and the



3. The 16-clock T_M period is generated when the multiplication op code is detected. FF2 provides an inverted

pulse that presets the counter to 0001. Unless otherwise indicated, TTL circuits and levels are implied.

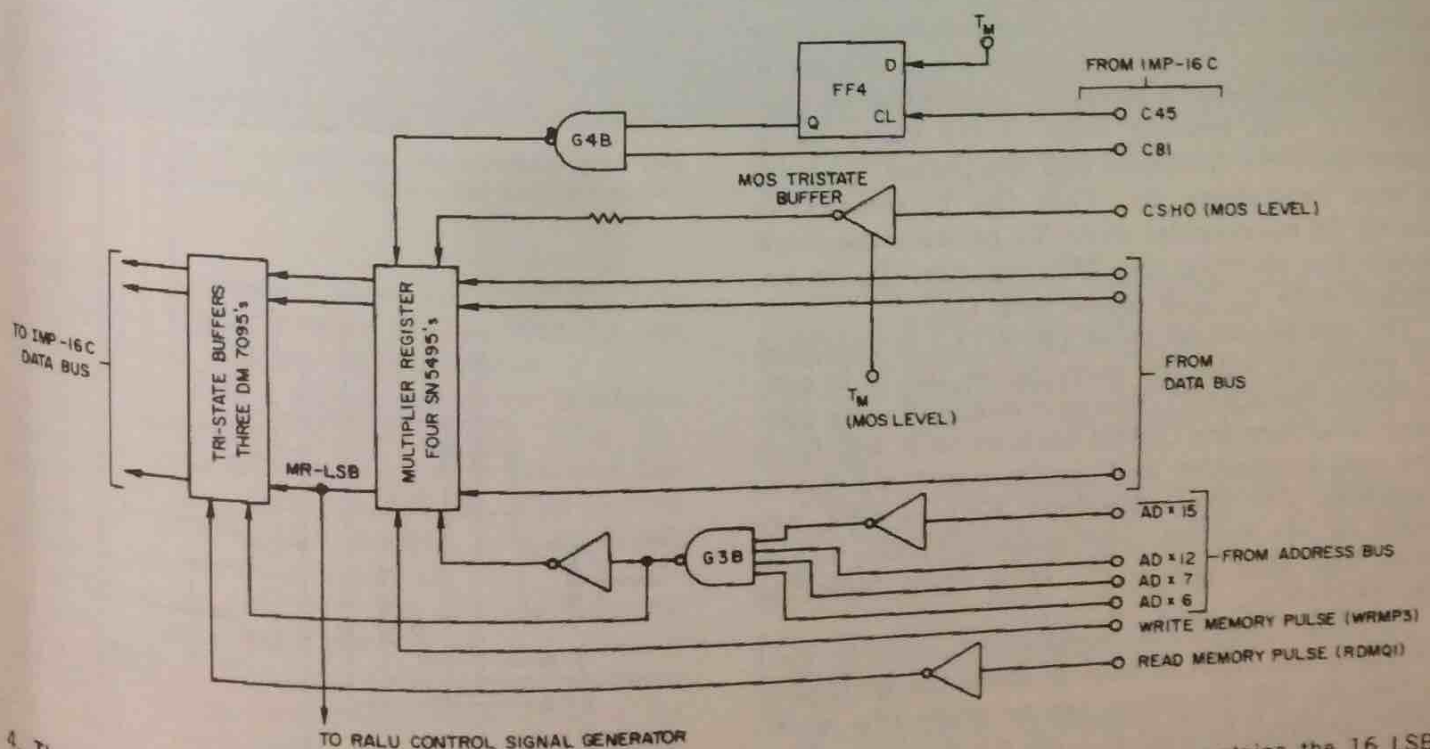
Table 3. Critical timing for the RALU

Signals		Logic levels	Time intervals								Pin function	Pin No.
			T ₁	T ₂	T ₃	T ₄	T ₅	T ₆	T ₇	T ₈		
Clocks ϕ_1 ϕ_2 ϕ_3 ϕ_4	MOS										IN	2
	MOS										IN	1
	MOS										IN	23
	MOS										IN	22
Command NCB(0) NCB(1) NCB(2) NCB(3)	MOS	$\overline{A0}$	"0"	$\overline{B0}$	"0"	$\overline{ALU0}$	"0"	$\overline{R0}$	"0"	IN	21	
	MOS	$\overline{A1}$	"0"	$\overline{B1}$	"0"	$\overline{ALU1}$	"0"	$\overline{R1}$	"0"	IN	19	
	MOS	$\overline{A2}$	"0"	$\overline{B2}$	"0"	$\overline{CTL0}$	"0"	$\overline{R2}$	"0"	IN	18	
	MOS	\overline{STACK}	"0"	\overline{COMP}	"0"	$\overline{CTL1}$	"0"	$\overline{I/O}$	"0"	IN	20	
Data DATA(0),(1),(2),(3)	TTL	R BUS (OUT)		A BUS(OUT)		"1" (OUT) ³		DATA INPUT	"1" (OUT)	I/O	17, 5, 4, 7	
Control FLAG	TTL	← FLAG →		"1"						OUT	16	
Misc SININ CSH0 CSH3	MOS T5 TTL T7	← Don't Care (DC) →				BYTE	DC	SIGN	DC	IN	10	
	MOS	"1" (OUT)	HIGH IMPEDANCE ²		CARRY (IN)	"1" (OUT)	\overline{SHIFT} I/O		I/O	14		
	MOS	\overline{OVCEN} (IN)	HIGH IMPEDANCE	"0" (OUT)	CARRY (OUT)	"1" (OUT)	\overline{SHIFT} I/O		I/O	11		

Note 1. A positive true logic convention is used for all signals—"1" = more positive voltage, "0" = more negative voltage. Signal names beginning with N are complemented signals.

Note 2. CSH0 and CSH3 high impedance states for intervals T₂ through T₄ are Tristate mode for output drivers.

Note 3. "1" (OUT) means RALU is driving this node to the "1" logic level during the defined interval. For bidirectional I/O lines the logic state is defined as "in" or "out."



4. The external multiplier register first stores the multiplier operand, then shifts it right 1 bit per microcycle, and then injects the least-significant partial product bits.

After 16 microcycles, the register contains the 16 LSB double-precision product. The multiplier register employs four 4-bit universal shift registers.

timing of all signals to and from the IMP-16C play a critical role in the hardware multiplication operation. The problem is aggravated by the fact that the speed required—a 6.5-MHz clock—is beyond the capabilities of most available standard SSI and MSI/MOS circuits.

To bridge this interface gap, one MOS-to-TTL and four TTL-to-MOS converters are needed, all of which must switch in less than 50 ns. The MOS-to-TTL conversion is relatively simple; it can be accomplished with a standard CMOS inverter. The TTL-to-MOS—+5 to -12 V—converters require capability to AND two TTL-input signals and to have an open-collector output that pulls low.

The NCBX line drivers present another interface problem. The signals must have MOS levels, and they must be connected to the bus only during T_M . Yet they must switch in less than 50 ns. The Motorola MC 14502 strobed hex inverter provides three-state output, the +5 to -12 V levels and the high speed.

The solution

The timing generator is initiated when the multiplication op code appears on the data bus (Fig. 3). Data bits 7, 10, 11, 12, 13, 14 and 15 are ANDed at the data input of a two-stage shift register (FF1 and FF2), which is clocked with the leading edge of clock period C_{s1} (waveform A). The Q output of FF2 is thus an inverted pulse, exactly 1 microcycle wide and starting 1 microcycle after the decoded op code is clocked into FF1 (waveform B).

The inverted pulse presets the 4-bit counter to 0001 and FF3 to $Q = 1$. Thereafter the counter increments with every C_{s1} pulse until count 15 is reached and a carry is generated (waveform C). The trailing edge of the carry pulse resets the FF3 Q output and the period T_M back to zero (waveform D). Thus the T_M interval is exactly 16 microcycles wide. To produce the clock pulses for shifting the MR register, T_M is re-clocked with C_{45} and gated with C_{s1} .

The enable control pulse (ENCTL) is produced by gating ϕ_3 with the carry pulse, but it is connected to the ENCTL line only during T_M . Its purpose is to turn the CROM back on so it will fetch the next instruction and continue with the macro program. The three flip-flops, FF1 to FF3, are reset by the system clear pulse, SYCLR, to ensure that they are in the reset state following power turn on.

A 16-bit parallel-in/parallel-out register that shifts to the right is used to store the multiplier (Fig. 4). Its functions are to store the multiplier operand, to shift it 1 bit right every microcycle and to shift the low-order product bit on the CSHO line into the register. After 16 micro-

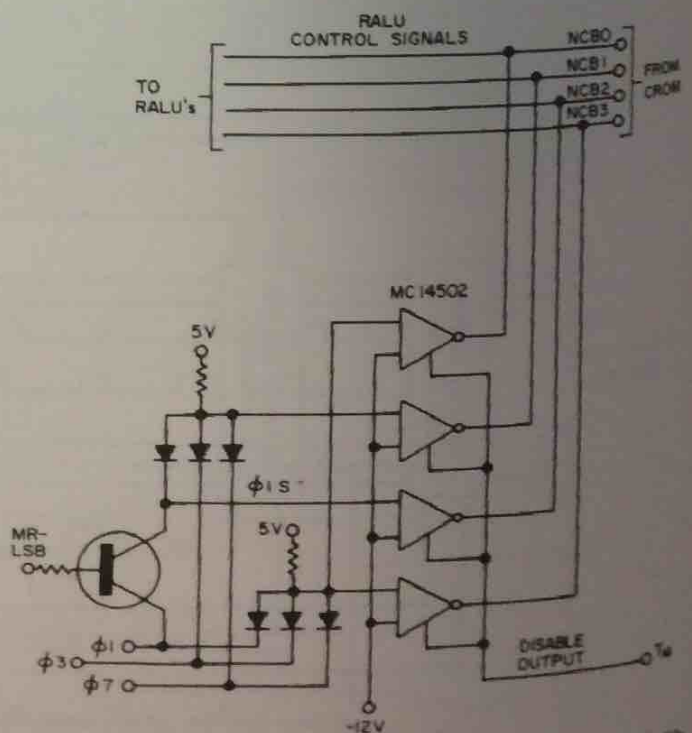
cycles, the register contains the 16 least-significant bits of the double-precision product.

The register employs four 4-bit universal shift registers and three Tristate hex buffers. The multiplier operand is loaded into the MR register prior to the actual multiplication operation (before T_M), with a macro STORE instruction that addresses the register as if it were another memory location. The use of specific memory addresses for peripheral devices has an advantage: The access is faster, and all memory-reference instructions can be used.

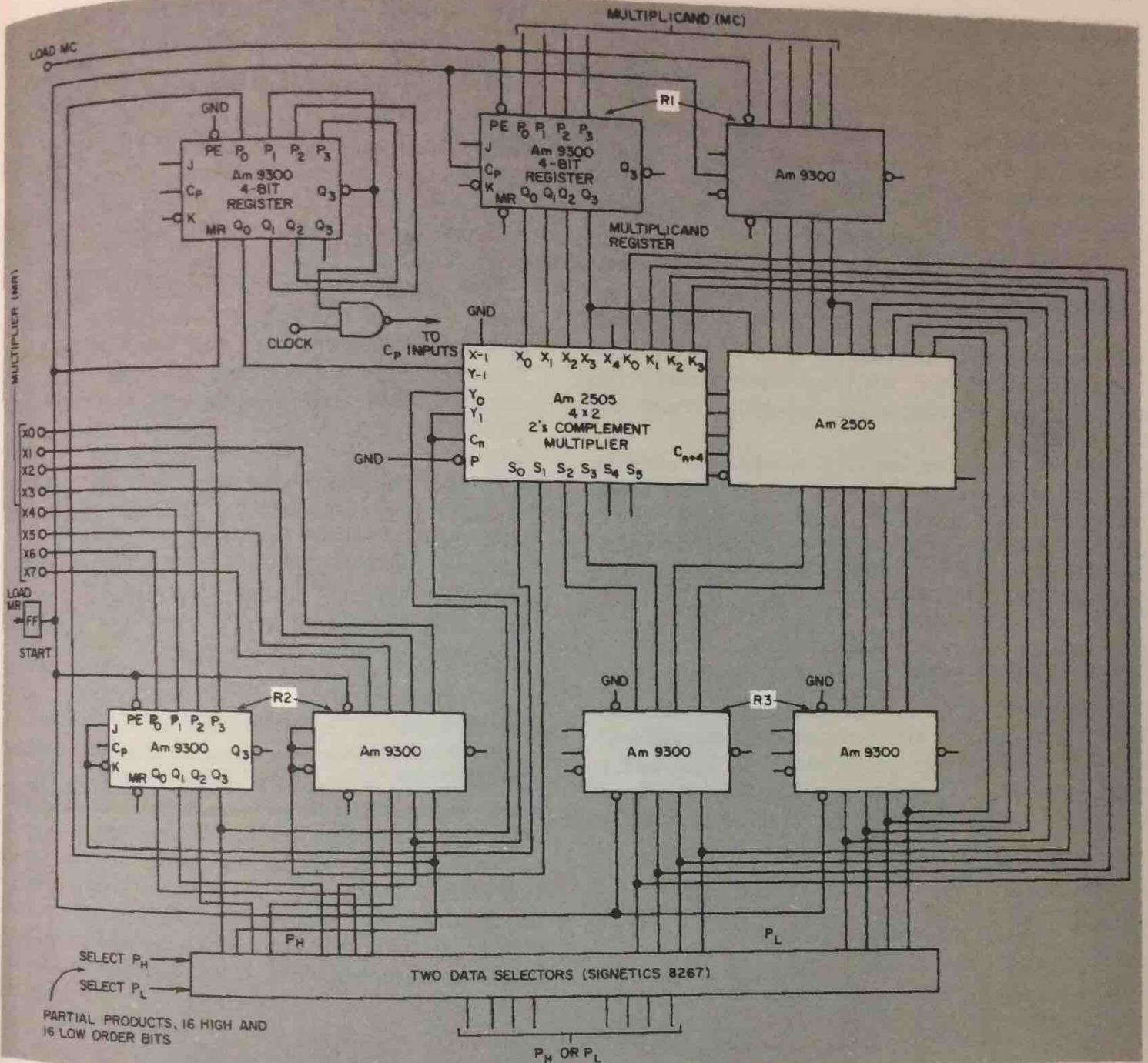
The output of the address-decoding gate (G3B) connects to the mode-control input (pin 6) of the four 4-bit shift registers. When the registers are addressed, the mode control signal is high, and when the clock-2 signal (WRMP3) switches from ONE to ZERO, the registers perform a parallel-load operation. This loads the multiplier operand into the MR register.

The gated clock signal at the G4B output shifts the MR register content 1 bit right at the leading edge of the C_{s1} timing pulse. The signal (MR-LSB) on the least-significant MR register output line thus constitutes the multiplier operand in serial-binary form.

The contents of the multiplier register must be loaded back into accumulator 1 of the RALU following completion of multiplication. For that purpose, a memory LOAD operation must be executed. The register is addressed as in the STORE operation. But when the read-memory pulse (RDMQ1) occurs, the MR contents are sent to



5. The control-signal generator produces pseudo micro-instructions that will execute the required add and shift operations in the RALU.



6. Two 4×2 -bit IC multipliers sequentially add the 8-bit parallel multiplicand into the partial product regis-

ter. Control of the operation is provided by the two least significant multiplier bits.

the data bus by a set of three noninverting Tri-state buffers. These buffers are enabled by the decoded address and the RDMQ1 pulse.

The RALU control signal generator produces four time-multiplexed NCBX signals (Fig. 5). The signals control the RALU, so it executes the required add and shift operations. The circuit consists of a transistor switch, two diode-OR gates and four strobed inverters.

The transistor connects the ϕ_1 clock pulse to the diode-OR gate when $MR - LSB = 1$. The six diodes form two three-input OR gates that combine the ϕ_3 and ϕ_7 clock pulses with either ϕ_1 or the switched clock pulse, ϕ_{18} . The strobed MOS inverters are Tristate devices that connect the NCBX signal to the RALU control bus only during T_M .

Though peripheral multipliers can be built with a number of available MSI ICs, only serial-parallel (rather than all-parallel) multipliers are cost and speed-compatible with microprocessors. The 4×2 -bit IC multiplier² in Fig. 6 is an example.

The 8×8 -bit peripheral multiplier consists of three single-length registers, a 2×8 -bit multiplier, an output data selector and some addressing and control logic. Register R₁ holds the multiplicand, while register R₂ holds the multiplier. Double-precision products are stored in registers R₂ and R₃.

At the start of multiplication, the multiplicand and multiplier are loaded into registers R₁ and R₂, respectively. During an operation the multiplier is shifted out, 2 bits at a time, and the

empty locations are filled with partial product bits.

The two Advanced Micro Devices multipliers provide a 2×8 -bit product during each clock cycle. Under control of the two least-significant multiplier bits, the multiplicand is added with appropriate weights to the eight LSBs of the partial product. The result is then stored again in the partial-product register.

The multiplier operand and the eight most-significant partial product bits are separated into odd and even parts, each of which is loaded into one 4-bit shift register. A shift of two places can thus be obtained with one clock pulse, with each register providing one least-significant bit as an output.

Four shift operations are needed to execute 8×8 -bit operations. This means that the basic multiplication execution time is four clock periods. Since these MSI circuits easily can operate at a clock rate of 4 MHz, the complete multiplication can be performed in 1 μ s.

Extending the technique

The technique can be extended easily to handle 16-bit multiplicand and multiplier operands and a 32-bit double-precision product. In the latter case, however, the length of all registers and of the actual multiplier must be doubled. Similarly

the execution time also increases by a factor of two, since eight shift operations must be performed now.

To use a peripheral multiplier in a microprocessor system, the multiplier must operate through the data bus and be treated like any other peripheral. When multiplication is performed, the microprocessor addresses a multiplier register and specifies whether the data are to be stored or fetched.

In a typical operation, the register would first store the multiplicand and multiplier operands and then fetch the high and low product bytes. This necessitates two output and two input operations, or four macroinstructions, and this would require much more time than the actual multiplication.

In the National IMP-16, each I/O operation requires approximately 10 μ s. Consequently the total multiplication execution time would be 40 μ s. And for a complete 16×16 -bit peripheral multiplier, with all the address decoding and control logic, approximately 18 MSI and eight SSI integrated circuits are required.

References

1. IMP-16C Application Manual, 4200021B, National Semiconductor Corp., Santa Clara, CA, 1973.
2. Ghest, R. C., "A Two's Complement Digital Multiplier," Application Note, Advanced Micro Devices Corp., Sunnyvale, CA, 1971.